計算機アーキテクチャ講義
ノート２
（62ページ～83ページ）

平成17年11月12日配布

今瀬　真

３．ハードウェア・アーキテクチャ

- 計算機の性能向上：1960年からずっと10年で10倍程度の性能改善
  - LSI技術など部品技術の進歩が寄与
  - 部品技術を活かすプロセッサ構成技術があって始めて実現可能
- 本章では、計算機のハードウェア構成を理解するとともに、いかに高速化の工夫がなされているかについて理解する。
  - 演算回路
  - 命令の実行制御（パイプライン制御）
  - メモリアクセス（キャッシュメモリ）

3．1（1）加減算回路

- 2の補数表現により、減算は加算に変換可能
- 1ビット加算器
  - $S_i = a_i \oplus b_i \oplus c_i$
  - $C_{i+1} = a_i \cdot b_i + (a_i + b_i) \cdot c_i$
  - ただし、⊕は排他的論理和、+は論理和　・は論理積

a_i	b_i	c_i	S_i	C_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1

a_i	b_i	c_i	S_i	C_{i+1}
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

[問13] 排他的論理和a⊕bおよびa⊕b⊕cを論理積・、論理和＋、否定~で表現せよ。

3．1（1）加減算回路

1ビット加算器

- $S_i = a_i \oplus b_i \oplus c_i$
- $C_{i+1} = a_i \cdot b_i + (a_i + b_i) \cdot c_i = g_i + p_i \cdot c_i$
- $p_i = a_i + b_i$ が1：下位からきたキャリーを上位ビットに伝播する。**桁上げ伝播信号**
- $g_i = a_i \cdot b_i$ が1：下位からのキャリーがなくとも上位ビットへのキャリーが生成される。**桁上げ生成信号**

a_i	b_i	c_i	S_i	C_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1

a_i	b_i	c_i	S_i	C_{i+1}
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

3．1（1）加減算回路

直列加算器:

- 遅延回路Dを利用して1クロック遅らせて次の桁のciとして加算。
- 加算する2数は下位桁から1クッロクに1ビットずつ送り込む。
- nビットの加算にnクロック必要。

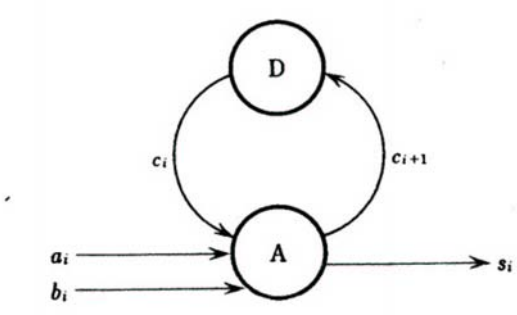


図 3.1 直列加算器（A：全加算器，D：遅延回路）

3．1（1）加減算回路

0110＋ 1011

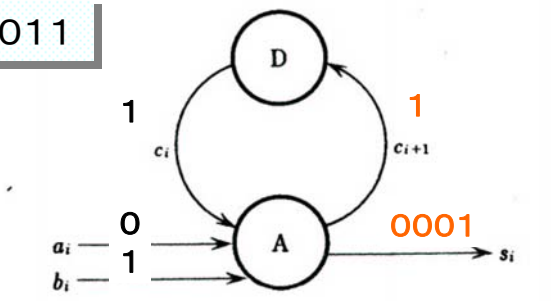


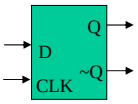
図 3.1 直列加算器（A：全加算器，D：遅延回路）

直列加算回路の実現

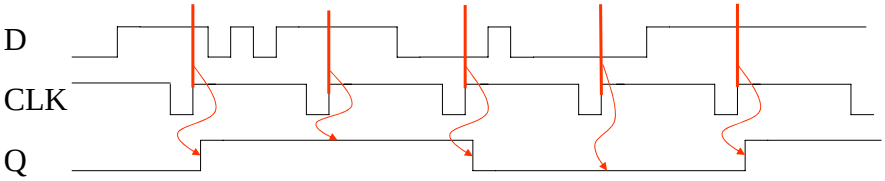
前述の直列加算器を実際にゲートで構成する場合、Dフリップフロップが必要となる。

- Dフリップフロップは、左記に示すように、クロックが立ち上がるときの入力を保持する。（下記に動作例を示す。
- これにより前述のDの部分が構成できる。

Dフリップフロップ



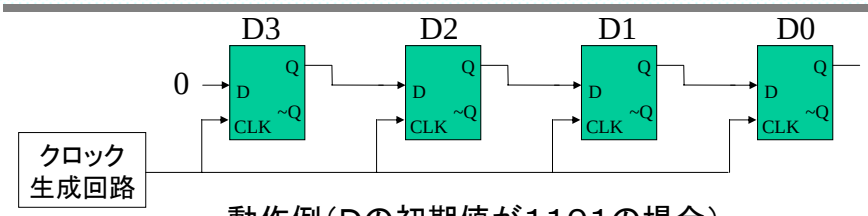
D	CLK	出力 Q	出力 -Q
1	↑	1	0
0	↑	0	1
X	その他	そのまま	そのまま



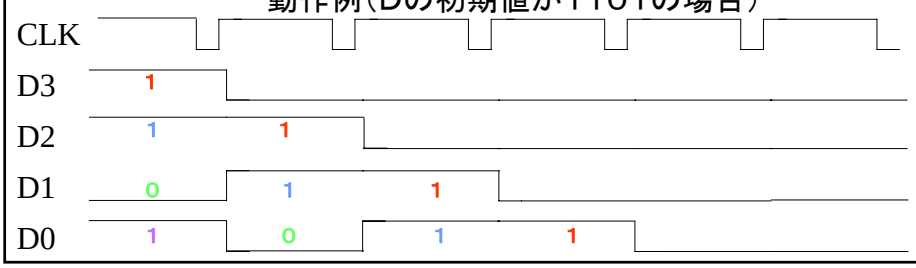
直列加算回路の実現

- 下記のようにDフリップフロップを並べると、クロックに合わせて1ビットずつシフトする回路が構成できる。

これにより前述のAへの入力部分が構成できる。答えの記憶も同様にできる。



動作例（Dの初期値が1101の場合）



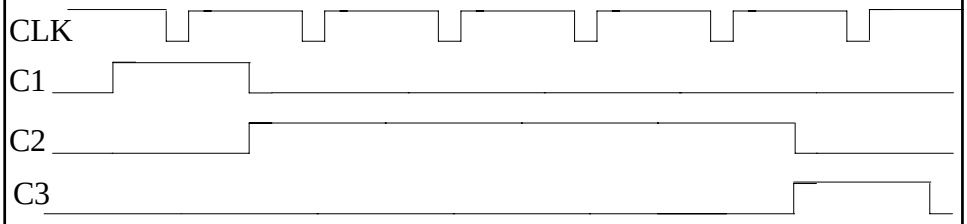
直列加算回路の実現

［問1］4ビットの直列演算回路を構成することを考える。ただし下記のことおよび次ページの全体構成を前提とする。次ページの機能ブロックであるシフトレジスタおよび加算回路をDフリップフロップ、論理和ゲート、論理積ゲート、否定を用いて構成せよ。

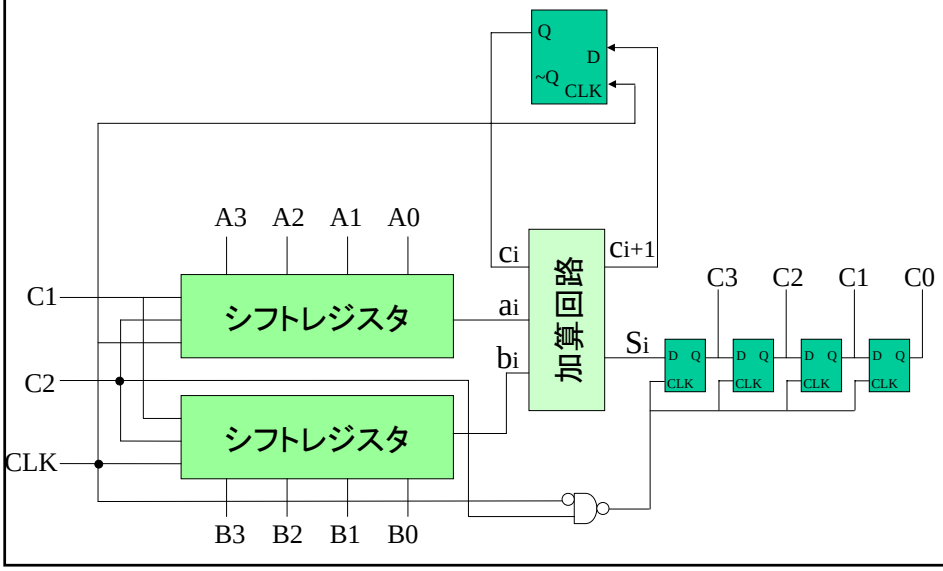
入力　A0,A1,A2,A3：被演算子Aが格納されているレジスタの出力
　　B0,B1,B2,B3：被演算子Aが格納されているレジスタの出力
　　C1：演算開始信号　C2：演算実行信号　C3：演算終了信号
　　CLK：クロック

出力　C0,C1,C2,C3：被演算子Aが格納されているレジスタの出力

演算開始/実行/終了信号とクロックのタイミング関係



問 1



3．1（1）加減算回路

順次桁上げ加算器：
- 下の桁の計算が終了しないと次のビットの桁の計算ができない。
- 各桁で2段のゲートを通過
- n桁で3nゲート分の遅延が発生（遅延はオーダn）
  →高速化をはかりたい。

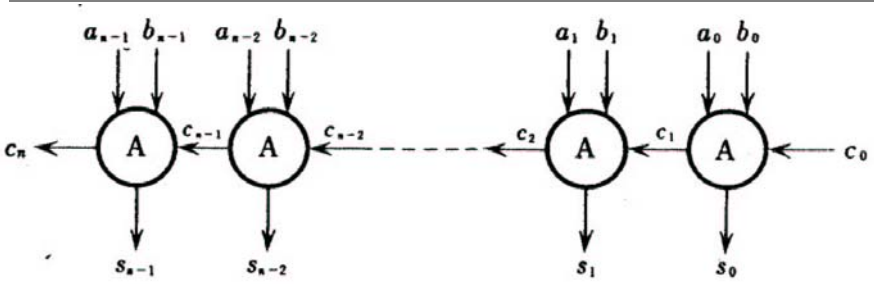
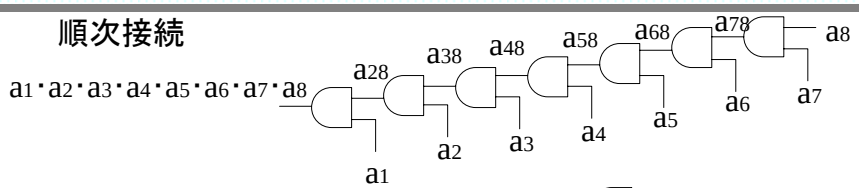


図 3.2 順次桁上げ加算器（A：全加算器）

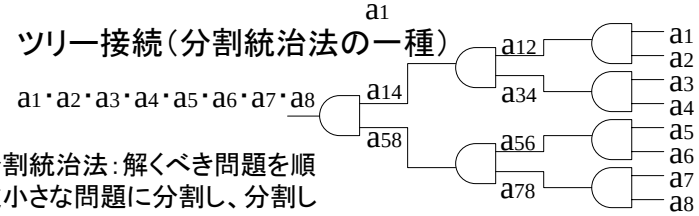
3．1（1）加減算回路

2分木構成による桁上げ先見加算器：準備
- 回路構成で順次接続とツリー接続でその計算時間が異なる
- 順次接続：n段、ツリー接続：log n段

順次接続



ツリー接続（分割統治法の一種）



分割統治法：解くべき問題を順次小さな問題に分割し、分割した小問題の解を統合して元の問題をとく

：論理積回路

3．1（1）加減算回路

1ビット加算器

- $S_i = a_i \oplus b_i \oplus c_i$
- $C_{i+1} = a_i \cdot b_i + (a_i + b_i) \cdot c_i = g_i + p_i \cdot c_i$
- $p_i = a_i + b_i$ が1：下位からきたキャリーを上位ビットに伝播する。**桁上げ伝播信号**
- $g_i = a_i \cdot b_i$ が1：下位からのキャリーがなくとも上位ビットへのキャリーが生成される。**桁上げ生成信号**

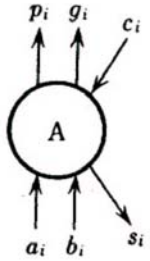
a_i	b_i	c_i	S_i	C_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1

a_i	b_i	c_i	S_i	C_{i+1}
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

3．1（1）加減算回路

2分木構成による桁上げ先見加算器1：

- 1ビットの加算器の出力を桁上げ信号c_{i+1}のg_iとp_iを出力
- g_iとp_iはa_iとb_iの関数でc_iを含まないことに注意



$p_i = a_i + b_i$ この桁でキャリーを伝播
$g_i = a_i \cdot b_i$ この桁でキャリーが生成
$S_i = a_i \oplus b_i \oplus c_i$

（a） 修正を加えた全加算器

3．1（1）加減算回路

2分木構成による桁上げ先見加算器2
（2ビットの加算器）

L内の回路

$P_i = p_{i+1} \cdot p_i$

桁i+1とiビットで桁上げ信号を伝播

p_{i+1}とp_i の関数でciを含まない

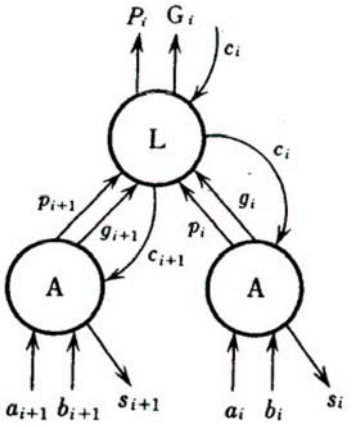
$G_i = g_{i+1} + g_i \cdot p_{i+1}$

桁i+1とiビットで桁上げ信号を生成

g_{i+1}、g_i、p_{i+1}の関数でciを含まない

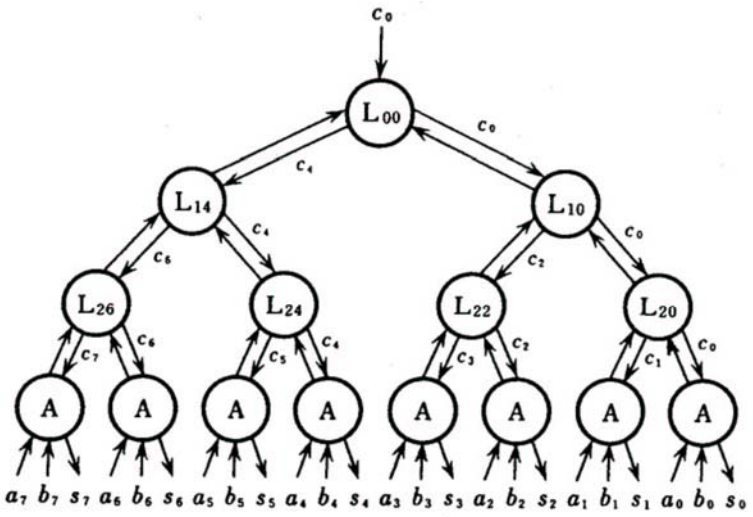
$c_{i+1} = g_i + p_i \cdot c_i$

- Lを多段に適用することが可能



（b） 2ビット桁上げ先見加算器

2分木構成による桁上げ先見加算器2（8ビットの加算器）



（c） 8ビット桁上げ先見加算器

図 3.3 2分木構成による桁上げ先見加算器

3．1（1）加減算回路

2分木構成による桁上げ先見加算器2（8ビットの加算器）

L_{10}:

$P_{10} = P_{22}P_{20} = p_3 \cdot p_2 \cdot p_1 \cdot p_0$

– 0～3ビットでc_0を4ビット目に伝播させるか否か

$G_{10} = G_{22} + G_{20} \cdot P_{22} = (g_3 + g_2 \cdot p_3) + (g_1 + g_0 \cdot p_1) \cdot p_3 \cdot p_2$

$= g_3 + g_2 \cdot p_3 + g_1 \cdot p_3 \cdot p_2 + g_0 \cdot p_3 \cdot p_2 \cdot p_1$

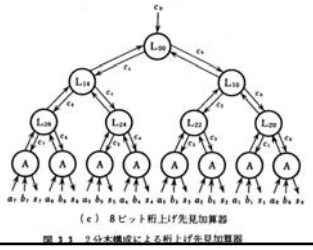
– 0～3ビットで4ビット目に伝播するあらたなキャリーが生成されたか否か

- 練習問題3．1と練習問題3．2をやっておくこと

•木の高さは$\log_2 n$

•最下位ビットで生じた桁上げはL_{10}からL_{20}を通過しL_{00}に登り、L_{14}、L_{26}で、を降り最上位ビットに到達する。

•速度は$2\log_2 n$に比例（オーダ $\log_2 n$）



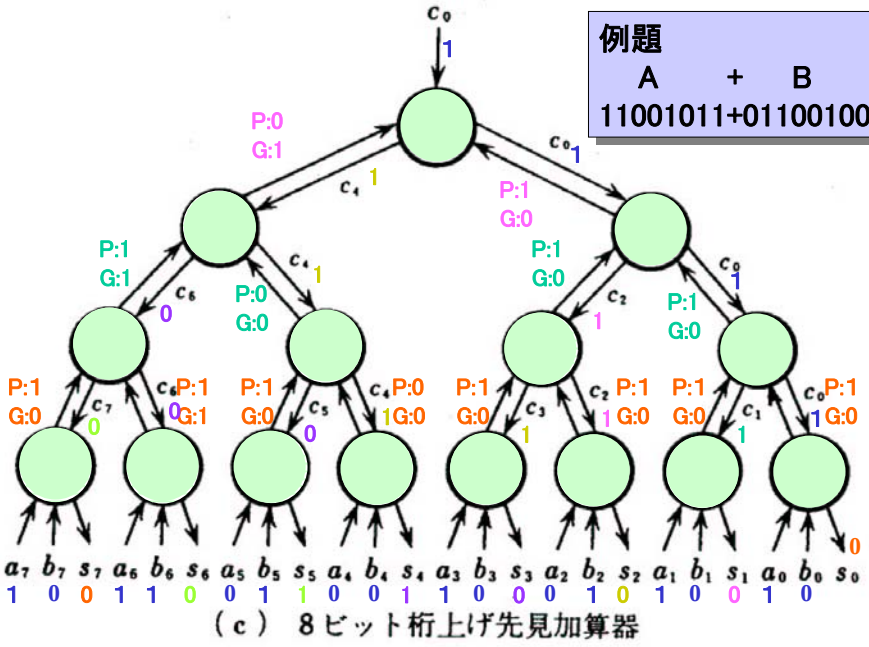


図 3.3 2分木構成による桁上げ先見加算器

3．1（1） 加減算回路

4分木構成による桁上げ先見加算器

- 桁上げ先見回路Lを4ビットに拡張
  - $P_i = p_{i+3} \cdot p_{i+2} \cdot p_{i+1} \cdot p_i$
  - $G_i = g_3 + g_2 \cdot p_3 + g_1 \cdot p_3 \cdot p_2 + g_0 \cdot p_3 \cdot p_2 \cdot p_1$
- 2分木構成では2入力1出力の論理回路を利用、4分木構成では4入力1出力の論理回路を利用でき、高速化が可能。

3．1（1） 加減算回路

- キャリースキップ加算器
- グループ化し（例は4ビットにグループ化）下位桁からきたキャリー信号を上位グループに高速に伝播する。Pだけを実現。

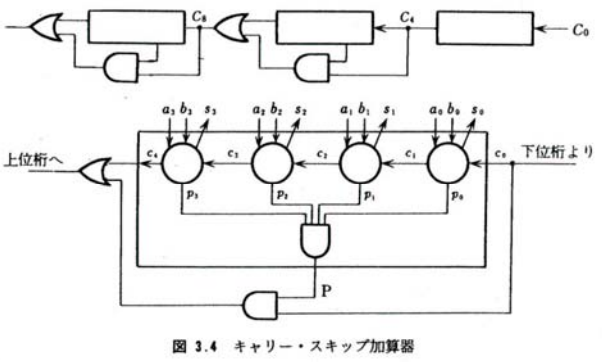


図 3.4 キャリー・スキップ加算器

•計算のオーダはn
•順次桁上げ加算器にくらべて定数部を改善

3．1 （1） 加減算回路

桁上げ保存回路（CSA）

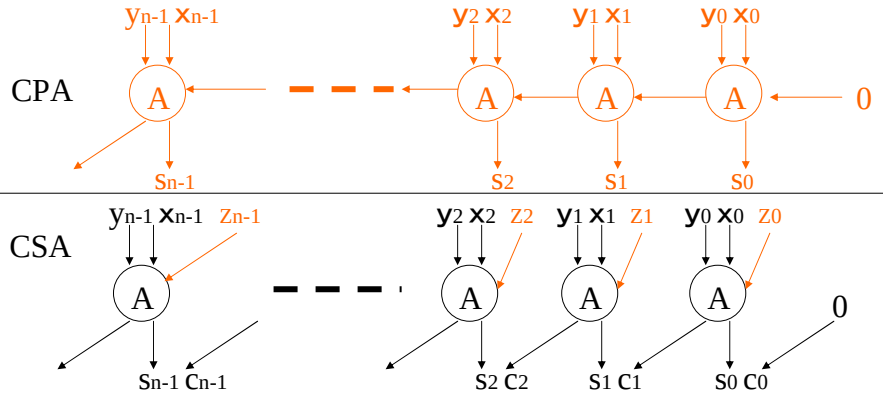
- CPAはA+B→Cを計算。CSAはX+Y+Z→S+Cを計算。（X+Y+Z＝S+Cを満たすSとCの組のひとつをもとめる）。
- CSAは1ビット加算器で容易に構成することができる。演算が高速（O（1））
- 複数オペランドの足し算を行うとき、CSA一つでオペランドの数が一つ減少する。
  - X1+X2+X3+X4+X5→X1+X2+S1+C1→S2+C2+C1 →S3+C3
- オペランド2つから1つにするにはCPAが必要
- 複数の数の足し算は、乗算回路をつくる時に必要

3．1 （1） 加減算回路

桁上げ保存加算器（CSA）

- 3つの2進数X=(x_{n-1},x_{n-2}・・・・x_0) Y=(y_{n-1},y_{n-2}・・・・y_0) Z=(z_{n-1}, z_{n-2}・・・・z_0)を入力として、2つの2進数S=(s_{n-1}, s_{n-2}・・・・ s_0) C=(c_{n-1},c_{n-2}・・・・ c_1 ,0)を出力
- 回路は全加算器と変わらない
  - $s_i = x_i \oplus y_i \oplus z_i$
  - $c_{i+1} = x_i \cdot y_i + (x_i + y_i) \cdot z_i$
- 前述した3つの加算器はこれに対比して、桁上げ伝播加算器CPAと呼ぶことがある。
- 元の3つの数X,Y,Zの和を求めるには、SとCの和を求める桁上げ伝播加算器が必要

3．1（1）加減算回路



桁上げ保存加算器（CSA）とCPAの比較

- ハード量　CSA：CPA＝1：1．5　程度
- 遅延時間　CSA：CPA＝1：5　　程度

3．1（1）加減算回路

並列加算器：CSAとCPAを併用することにより高速化を実現可能

- 練習問題3．3は理解しておくように

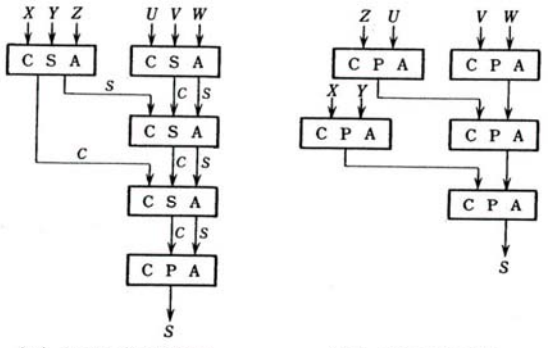


図 3.5　6 入力の並列加算

3．1 （2） 乗算回路

- nビットの2つの数の乗算結果は最大2nビット（2の補数の時は2n−1）になる。
- 下記の議論は、2つのnビットの数A,Bから2nビットの乗算結果を求める問題について述べている。
- 計算機のハードウェアを構成する場合は、結果がnビットを超えた場合はオーバフロービットをたてるとか倍精度に変換するなどの処理が必要となる。ここでは、その問題には触れない。
- 2の補数系の乗算回路を考える上で基本となるのが式（3.11）

$$A \times B = -a_{n-1}B2^{n-1} + a_{n-2}B2^{n-2} \cdots\cdots a_1B2^1 + a_0B2^0$$

A×Bの証明（補足説明）

$$A \times B = -a_{n-1}B2^{n-1} + a_{n-2}B2^{n-2} \cdots\cdots a_1B2^1 + a_0B2^0$$

となることの証明の方針

- 2の補数系は下記のことが成立する。

$x_{n-1}=0$の時 $(X)_{2C} \equiv (X)_2$、$x_{n-1}=1$の時 $(X)_{2C} \equiv (X)_2 - 2^n$

すなわち $a \geqq 0$の時 $I_{2C}(a) \equiv I_2(a)$、 $a < 0$の時 $I_{2C}(a) \equiv I_2(a + 2^n)$

- AとBの正負について場合分け（4通り）して、上記の式が成立することを証明する。（正負の区別は最上位ビットが1か0で行える。）
- 詳細はレポート課題

3．1 （2） 乗算回路

逐次型乗算器（図3．6）

- 図中、上のレジスタは2n+1ビット、下のレジスタはnビット
- 加算器：2入力のnビットデータを加算しn+1ビットの結果を出力

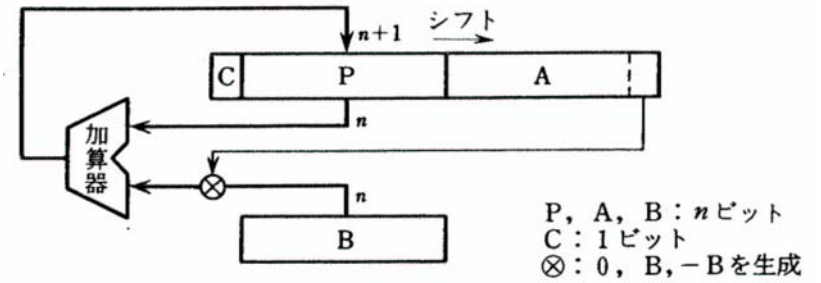


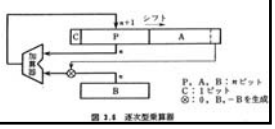
図 3.6 逐次型乗算器

3．1 （2） 乗算回路

逐次型乗算器（図3．6）の計算手順

- 初期設定：上のレジスタの上位n+1ビットに0、下位nビットにA、下のレジスタにBを代入。以下の3つの動作をi=0～n−1繰り返す。
- 加算器の下側に下記の値を出力
  - ai=0の時：0
  - ai=1かつi≠n−1の時：B　ai=1かつi=n−1の時、－B
- 加算器で和を計算
- 加算結果を上のレジスタの上位n+1ビットに代入
- 上のレジスタを1ビット右にシフト

上のレジスタの下位2nビットが求める結果

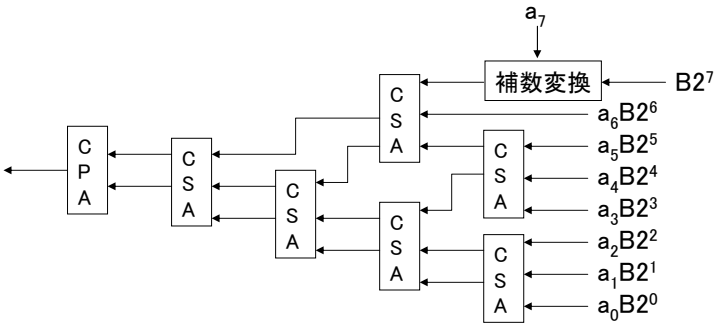


3．1 （2） 乗算回路

並列乗算器

$A \times B = -a_{n-1}B2^{n-1} + a_{n-2}B2^{n-2} \cdots\cdots a_1B2^1 + a_0B2^0$

- 補数変換、CSA（n−2個）、CPA（1個）を組み合わせることにより実現
- 32ビットの乗算では、レベル数はCPAを含めて9



3．1 （2） 乗算回路

- ブースのアルゴリズムについては省略
- 乗算回路のゲート数と演算時間

		計算時間	回路量
加算	直列加算器	$O(n)$	$O(1)$
	順次桁上げ加算器	$O(n)$	$O(n)$
	2分木構成による桁上げ先見加算器	$O(\log n)$	$O(n)$
	キャリースキップ加算器	$O(n)$	$O(n)$
	桁上げ保存加算器CSA	$O(1)$	$O(n)$
乗算	逐次型加算器（加算に先見加算器を使った時）	$O(n \log n)$	$O(n)$
	並列型加算器	$O(\log n)$	$O(n^2)$

3．1 （3） 除算回路

- 乗算回路のように正の数と負の数を統一して扱う方法はない。
- 以下ではnビットの正整数の除算について説明する。
- 簡単のため、被除数X、除数Dをnビットの符号なしの正の（1以下の）少数、被除数X＜除数Dとする。

$$X = Q \cdot D + R \quad 0 <= R < D \cdot 2^{-n}$$

をみたすQを求めればよい

- $R_i = 2R_{i-1} - q_i \cdot D \quad 0 <= R_i < D$ （ただし$R_0 = X$）

により逐次q_i =0または1を求めQ=0. $q_1 q_2 q_3 \ldots q_n$とすれば、

$Q \cdot D = D \cdot (q_1 2^{-1} + q_2 2^{-2} + \ldots\ldots + q_0 2^{-n})$

$= (2R_0 - R_1) 2^{-1} + (2R_1 - R_2) 2^{-2} \ldots\ldots + (2R_{n-1} - R_n) 2^{-n}$

$= R_0 - R_n 2^{-n} = X - R_n 2^{-n}$

余り$R = R_n 2^{-n}$とすればQとRが求める解になる。

3．1 （3） 除算回路

- $R_i = 2R_{i-1} - q_i \cdot D \quad 0 <= R_i < D$ （ただし$R_0 = X$）

により逐次q_i =0または1を求めQ=0. $q_1 q_2 q_3 \ldots q_n$とすれば、

$Q \cdot D = R_0 - R_n 2^{-n} = X - R_n 2^{-n}$

余り$R = R_n 2^{-n}$とすればQとRが求める解になる。

例： $R_0 = X = 0.100_2$ $D = 0.110_2$

$R_1 = 2R_0 - q_1 \cdot D = 1.000_2 - q_1 \cdot 0.110_2 \rightarrow q_1 = 1 \; R_1 = 0.010$

$R_2 = 2R_1 - q_2 \cdot D = 0.100_2 - q_2 \cdot 0.110_2 \rightarrow q_2 = 0 \; R_2 = 0.100$

$R_3 = 2R_2 - q_3 \cdot D = 1.000_2 - q_3 \cdot 0.110_2 \rightarrow q_3 = 1 \; R_3 = 0.010$

Q

3．1 （3） 除算回路

引き戻し法

- 上のレジスタは2n＋1ビット、下のレジスタはn+1ビット
- 初期化
  - 上のレジスタ：上位1ビットは0、2ビット～n+1ビットにR、n+2ビット～2n+1ビットは0を代入
  - 下のレジスタ：上位1ビットは0 、2ビット～n+1ビットにDを代入

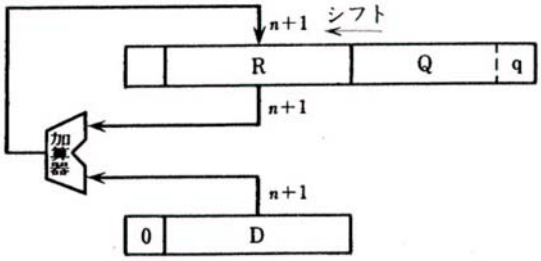


図 3.7 逐次型除算器

3．1 （3） 除算回路

引き戻し法の手順

1. 上のレジスタを左へ1ビットシフト（$2R_{i-1}$を上位n+1ビットに生成）
2. R←R−Dとし、Rが負ならが3へ、正ならば4へ進む
3. q(上のレジスタの最下位ビット)←0、R←R＋Dとして1へ
4. q(上のレジスタの最下位ビット)←1

最悪2n回の加減算が必要

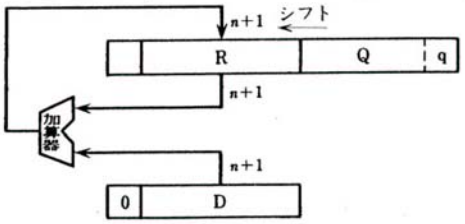


図 3.7 逐次型除算器

3．1 （3） 除算回路

引き放し法の手順

1. 上のレジスタを左へ1ビットシフトし、負ならば2へ、正ならば3へ
2. q←−1 R←R＋Dとし、1へ
3. q←1、R←R−Dとして1へ

n回の加減算で済む

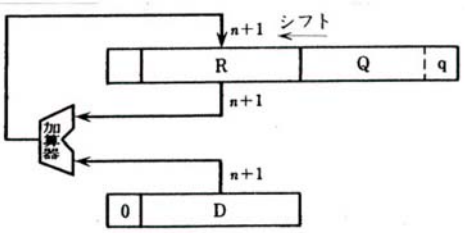


図 3.7 逐次型除算器

3．1 （3） 除算回路

例

$X = r_0 = .100_2 = (1/2)_{10}$

$D = .110_2 = (3/4)_{10}$

(引き戻し法)	R	$Q\ q$
(1) $2r_0$	01.000	00
$-D$	11.010	
r_1	00.010	001
(2) $2r_1$	00.100	01
$-D$	11.010	
	11.110	01
$+D$	00.110	
r_2	00.100	010
(3) $2r_2$	01.000	10
$-D$	11.010	
r_3	00.010	101

$Q = .101_2 = (5/8)_{10}$

$R = r_3 \cdot 2^{-3} = .000010_2 = (1/32)_{10}$

(引き放し法)	R	$Q\ q$
(1) $2r_0$	01.000	00
$-D$	11.010	
r_1	00.010	001
(2) $2r_1$	00.100	01
$-D$	11.010	
r_2	11.110	011
(3) $2r_2$	11.100	11
$+D$	00.110	
r_3	00.010	11-1

$Q = .11\text{-}1$

$= .110_2 - .001_2$

$= .101_2 = (5/8)_{10}$

$R = r_3 \cdot 2^{-3} = .000010_2 = (1/32)_{10}$

3．1 （3） 除算回路

- 乗算は並列化できるが除算は並列化する手法はしられていない。
- 計算オーダはnlogn
- 乗算を用いて除算を実現する手法が考案された

3．1 （3） 除算回路

乗算を用いた除算

- D:除数、X：被除数、Qを商　$D \cdot R_0 \cdot R_1 \cdot R_2 \cdot \ldots \cdot R_m$が1に収束するならば

  $Q = X \div D = X \cdot R_0 \cdot R_1 \cdot R_2 \cdot \ldots \cdot R_m \div (D \cdot R_0 \cdot R_1 \cdot R_2 \cdot \ldots \cdot R_m)$

  $= X \cdot R_0 \cdot R_1 \cdot R_2 \cdot \ldots \cdot R_m$
- 整数よりも浮動小数点の除算に適している。
- 簡単のため$\frac{1}{2} <= D < 1$ (D=0.1…)と仮定する。

1. $D_0 = D = 1 - y$, $0 < y <= 1/2$　　$R_0 = 1 + y$

   とすれば、$R_0 = 2 - D_0$　　$D_1 = D_0 \cdot R_0 = (1-y)(1+y) = 1 - y^2$

   となり、$y^2 <= 1/4$, $D_1 >= 3/4$ (D_1=0.11…)
2. $D_1 = 1 - y^2$, $0 < y^2 <= 1/4$　　$R_1 = 1 + y^2$ →$R_1 = 2 - D_1$　$D_2 = D_1 \cdot R_1 = (1 - y^2)(1 + y^2) = 1 - y^4$

   となり、$y^4 <= 1/16$, $D_2 >= 15/16$ (D=0.1111…)

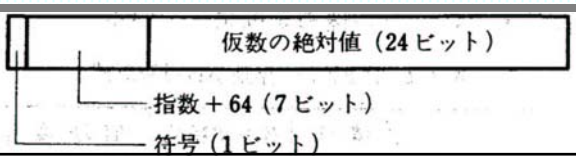
3．1 （3） 除算回路

乗算を用いた除算

- nステップ繰り返す。すなわち
  - $R_{i-1} = 2 - D_{i-1}$　　$D_1 = D_{i-1} \cdot R_{i-1}$
- これにより、$D_1 = 1 - y \cdot y \ldots\ldots y$ （ yの2^i乗）<= yの-2^i乗
- 即ち、$D_i = 0.1 \ldots 1$（1が2^i個ならぶ）。
- すべての桁が1なら$D_i = 1$に最も近い近似値が得られる。
- nビットの除算では、2logn回の乗算が必要（除算時間は$(\log n)^2$

3．1 （4） 浮動小数点演算回路

- 整数演算とシフトの組み合わせに帰着
- 加減算
  1. 加数と被加数の指数の差を求める。
  2. 大きい方の指数を答えの仮の指数とする。
  3. 指数が小さい法の(被)加数を指数の差だけ右にシフトする
  4. 加数と被加数の仮数部分の加減算を行う。
  5. 答えの正負の符号を定める。
  6. 演算結果(仮数)を正規化に必要なシフト桁数を求める。
  7. 仮数をシフトするとともに丸めを行う。
  8. シフト数だけ仮の指数を補正する。



3．1（4）浮動小数点演算回路

加減算の例（$0.1110*2^5+0.1111*2^3$）

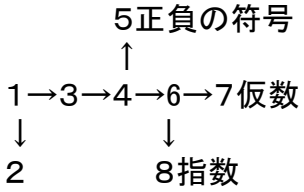
1. 指数の差を求める。5−3→2
2. 大きい方の指数を答えの仮の指数とする。$0.xxxxxx*2^5$
3. 仮数のシフト。$0.1111*2^3$を2ビット右シフト→$0.001111*2^5$
4. 仮数の加減算。0.1110＋0.001111→1.000111
5. 答えの正負の符号を定める。正
6. 正規化に必要なシフト桁数。 +1
7. 仮数のシフトと丸め。1.000111を1ビット右シフト→0.1000
8. 仮の指数を補正。5 +1 →6

$0.1000*2^6$

3．1（4）浮動小数点演算回路

- 加減算
  1. 加数と被加数の指数の差を求める。
  2. 大きい方の指数を答えの仮の指数とする。
  3. 指数が小さい法の(被)加数を指数の差だけ右にシフトする
  4. 加数と被加数の仮数部分の加減算を行う。
  5. 答えの正負の符号を定める。
  6. 演算結果(仮数)を正規化に必要なシフト桁数を求める。
  7. 仮数を左にシフトするとともに丸めを行う。
  8. シフト数だけ仮の指数を補正する。

並列処理化



3．1（4）浮動小数点演算回路

- 乗除算（例：$0.1110*2^5 \times 0.1111*2^3$）
  1. 仮数の乗除算を行う。0.1110×0.1111→1.1010010
  2. 指数の加減算を行う。5＋3→8
  3. 答えの正負の符号を定める。正
  4. 結果に応じて仮数を1ビット左シフトするとともに丸めを行う。
     1.1010010を1ビット左シフト→0.1101
  5. 結果に応じて指数を1だけ補正する。8+1→9

  $0.1101*2^9$

3．1（4）浮動小数点演算回路

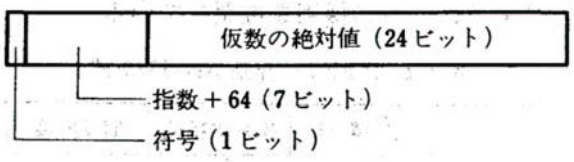
- 乗除算
  1. 仮数の乗除算を行う。
  2. 指数の加減算を行う。
  3. 答えの正負の符号を定める。
  4. 結果に応じて仮数を1ビット左シフトするとともに丸めを行う。
  5. 結果に応じて指数を1だけ補正する。

並列化

1→4仮数　3正負の符号

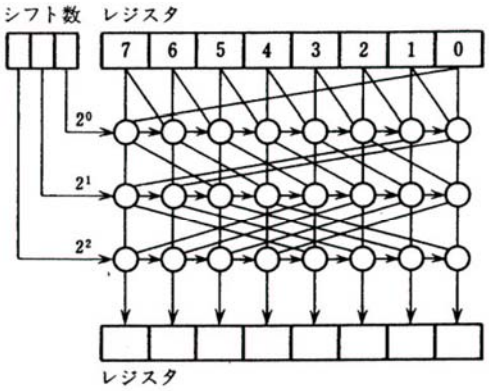
↓

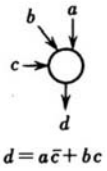
2→5指数



3．1 （4） 浮動小数点演算回路

- 浮動少数点の加減算では、桁合せのために任意ビットのシフト操作が必要
- バレルシフタが一般的（計算時間O(log n)で回路量O(nlog n)）





5ビットシフト→1ビットシフト＋4ビットシフト
3ビットシフト→1ビットシフト＋2ビットシフト
6ビットシフト→2ビットシフト＋4ビットシフト

図 3.8 バレル・シフタ

3．1 （5） マシンサイクルと演算回路のパイプライン化

論理回路：0と1の世界でAND、OR、NOT、NANDなどのゲートからなる
ゲート遅延：ゲートでは入力から出力がでるまでに必ず遅延が発生する。
組合せ論理回路：入力が決まると出力が一意に決まる。
順序論理回路：現在の出力が入力だけでなく、過去の履歴に依存する。（記憶素子が含まれている）

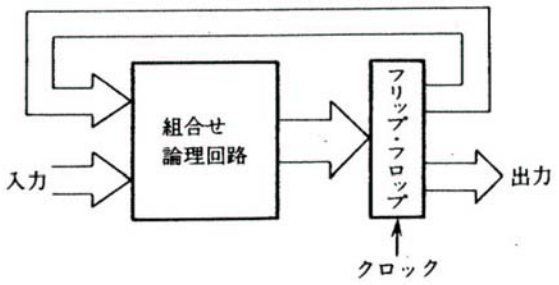


図 3.9 同期式計算機のモデル

3．1 （5） マシンサイクルと演算回路のパイプライン化

- マシンサイクル：CPUのあるクロックからつぎのクロックまでの期間を1マシンサイクルとよぶ。

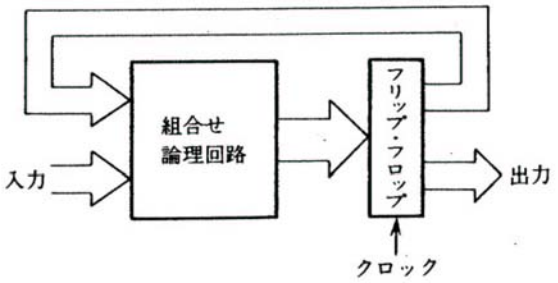
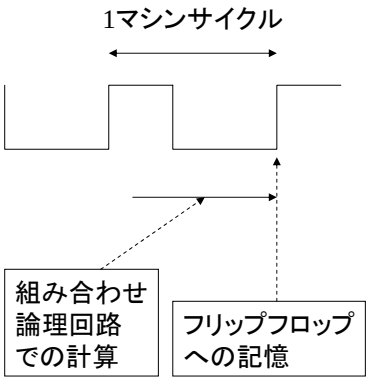


図 3.9 同期式計算機のモデル



3．1 （5） マシンサイクルと演算回路のパイプライン化

- レジスタと状態情報はフリップフロップの集合でその他は組合せ論理回路
- 1マシンサイクルの間にレジスタの出力が、演算回路(ALU)や様々のゲートを通り、再びレジスタなどの入力側に達し、次のクロック立ち上がりでレジスタ等が入力信号に応じた新しい値に設定される。

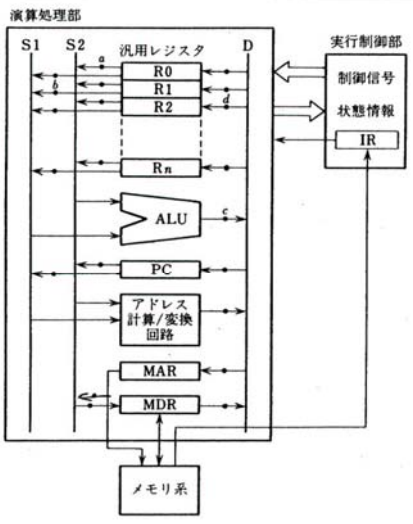


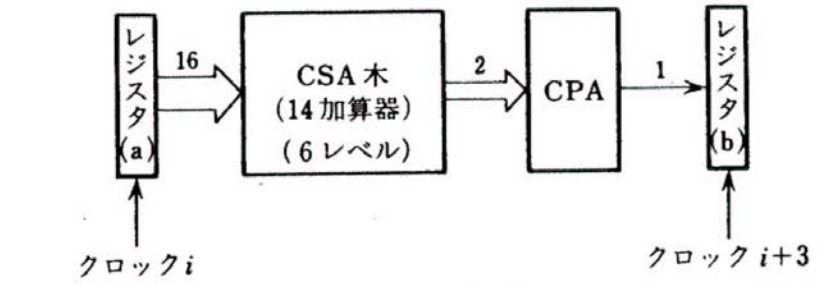
図 3.12 演算処理部と実行制御部（●印は制御ゲートの位置を示す）

3．1 （5） マシンサイクルと演算回路のパイプライン化

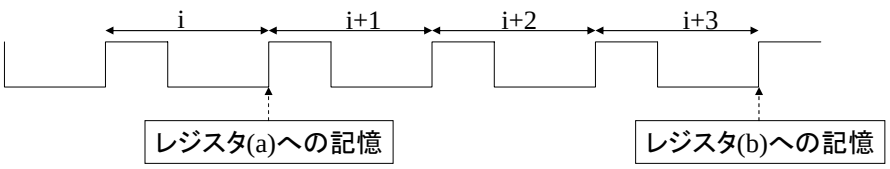
- 1マシンサイクルは組み合わせ論理回路の遅延に応じて設定する必要がある。（通過ゲート数が多いほど1マシンサイクルは長くなる。）
- 演算の種類によって通過ゲート数が異なる。通常は整数の加減算などの簡単な演算が1マシンサイクルで実現できるようにマシンサイクルを設定。（乗除算などは複数マシンサイクルが必要）
- 経験則から1マシンサイクルあたり20～30段のゲートでの演算遅延で設計

3．1 （5） マシンサイクルと演算回路のパイプライン化

- 3サイクル乗算回路の例1

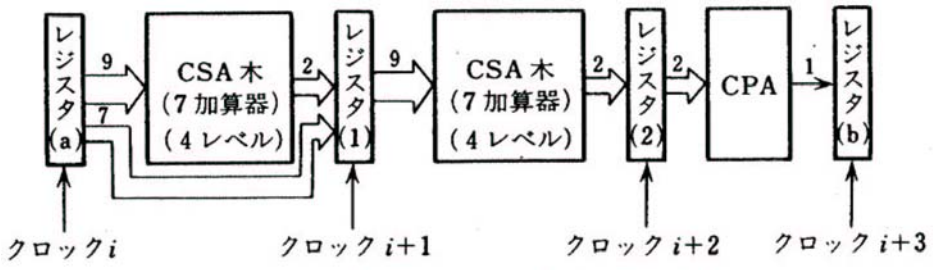


（a） 3サイクル乗算回路



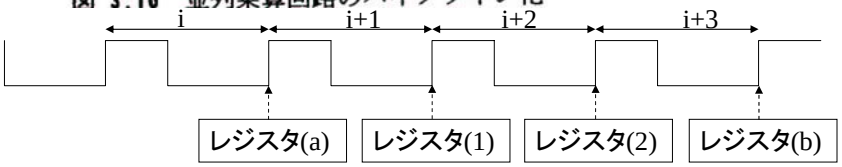
3．1 （5） マシンサイクルと演算回路のパイプライン化

- 3サイクル乗算回路の例2（パイプライン化）



（b） 3ステージ・パイプライン乗算回路

図 3.10 並列乗算回路のパイプライン化



3．1 （5） マシンサイクルと演算回路のパイプライン化

- レジスタ挿入の利点の利点：レジスタ(a)に1クッロクごとにデータを送れば、1クロックごとに結果をえることができる。
- 連続して演算を行うとすると、見かけ上の演算速度は1マシンクロックタイムとなる。
- これをパイプライン演算回路という。

	ｸﾛｯｸ1	ｸﾛｯｸ2	ｸﾛｯｸ3	ｸﾛｯｸ4	ｸﾛｯｸ5	ｸﾛｯｸ6	ｸﾛｯｸ7
データ1	ﾚｼﾞｽﾀ(a)→ﾚｼﾞｽﾀ(1)→ﾚｼﾞｽﾀ(2)→ﾚｼﾞｽﾀ(b)						
データ2		ﾚｼﾞｽﾀ(a)→ﾚｼﾞｽﾀ(1)→ﾚｼﾞｽﾀ(2)→ﾚｼﾞｽﾀ(b)					
データ3			ﾚｼﾞｽﾀ(a)→ﾚｼﾞｽﾀ(1)→ﾚｼﾞｽﾀ(2)→ﾚｼﾞｽﾀ(b)				
データ4				ﾚｼﾞｽﾀ(a)→ﾚｼﾞｽﾀ(1)→ﾚｼﾞｽﾀ(2)→ﾚｼﾞｽﾀ(b)			
・							
・