遅延線を用いるブロック演算について*

穂　坂　　衛**

1. まえがき

電子計算機研究の一つの方向は，より高速で，より flexible な機械を作ることで，記憶装置，演算装置，演算素子に高速で信頼性の高いものを開発すると同時に，システムの設計では並列の演算，同時に多くの独立の制御や先回りの制御などが考えられ，また機械の製造方法にも新しい方式が導入されようとしている．このように大形機の性能をより向上させることも大切であるが，中形，小形のものでも効率のよい機械を安価に作り，広く普及させることもまた重要である．従来の小中形機においては，磁気ドラムを記憶装置に用い，動作をすべて直列とし，clock も 100～200 kc 程度にして，全体の価格を下げてきたように思える．このような機械では，serial operation と access time の大きいことのために，すべての動作が，一様に能率が悪く，大形機にくらべ，価格の比よりも速度が低下するのは止むを得ないことであった．

筆者は，serial な general purpose computer を使用してきた経験(1)と serial な動作を主とした国鉄の座席予約装置の開発の経験(2)から一般向の電子計算機に対して次のような考えを抱くに至った．現在では，安価な信頼性ある機械を作ろうとすれば，記憶装置，演算装置などは serial なものを用いざるを得ない．general purpose の計算機の動作は，あらゆる場合に対処できる必要はあるが，実際に多く解くことを要求されている問題は，かなり限られた形式のものである．したがって，その種類の問題に対して高速な動作を行うことができれば，小形計算機であっても，実用上かなり能率があがり，また速度の点から小形機にかけることをあきらめていた種類の問題をも解くことができるであろう．たとえば，始めの種類の問題は大きなベクトルやマトリックスに関する問題，あとの問題では sorting や table look up がひんぱんに入り組んでくる問題などであり，いずれも多くのデータが演算の対象となり，繰返し回数が非常に多くなるものである．一般に stored program の計算機では，上述のような問題を処理するに当っては，address modification の技術，特に index register をそなえることによって，プログラムを効果的に作り上げることができる．

しかし，目的の四則演算に較べて，いわゆる bookkeeping や red tape 操作である address modification, initialization, termination, counting, branch などの操作が多ければ能率的な演算の進め方とはいえない．また逆に bookkeep や red tape の operation の割合が僅かである場合は index register が hardware としてなくても非常に演算時間が長くなることはないであろう．

一般に address modification の対象となるデータは規則正しい addres にあることが必要であるから，このようなデータを処理するに当って，その処理の仕方が簡単であるならば，命令を記憶装置からよみ出し，つぎにそれに指定されているデータを，記憶装置から読んでは演算を加え，また次の命令を読むというサイクルを繰り返しながら，プログラムを進めていくのは能率的でない．ここで subroutine に頼らず，一つの組みこみ命令で，規則的に配列されたデータのすべてに，定められた演算をほどこすことができたならば，演算速度の向上とプログラム作製の簡易化が達成できるであろう．このような一群のデータに対して操作を中絶しないで演算をほどこすことをブロック演算(blok operation)という．

ブロック演算が能率よく行われるためには，データはつぎつぎに記憶装置より読み出され，serial に処理され，また再び記憶装置にもどすことができ，データの流れが中断されないことである．このような操作に最も都合のよい記憶装置は，自然にデータがつぎつぎに読み出されて，またもとに書き込まれていく遅延線の記憶方式である．これでは，情報は常に再生されては循環している．ブロック演算では，データ処理の回路を遅延線と遅延線の間に入れることによって，流れていくデータが自動的に処理されていくのである．多くの計算機においては，記憶部と演算レジスタとは判然と区別され，記憶装置よりデータをレジスタに呼び

---

* On Block Operations Using Delay Lines, by Mamoru Hosaka

** 東京大学航空研究所

出して後，必要な処理操作が加えられ，結果をレジスタに残し，再びつぎの動作でそれを記憶装置にもどすというやり方をするのであるが，これでは命令の読み出し，解読，実行という動作を繰り返し，その間に，さらに命令変更の手続のための演算を行なわなければならない．

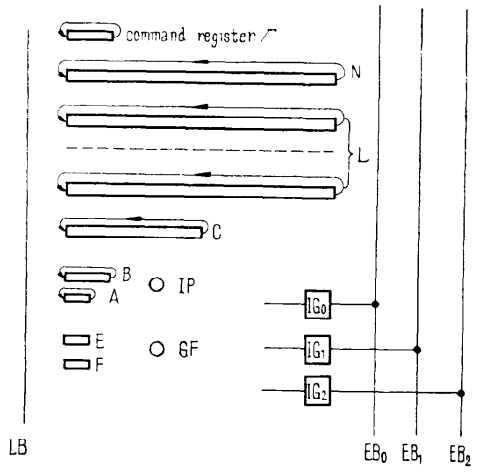
最近の大形の機械では bookkeep や red tape の operation はできるだけ能率よく，しかも主演算と平行して行うように考えられてきたが，ここで考えたブロック演算方式は命令の読み出し，実行の繰返しでなく，一連のデータを連続的に，時には同時操作を行って高速処理し，しかも hardware 自体を大がかりのものにしないということが大きな特徴である．

この方式に用いる遅延線には制限がない．ただ筆者がいままで実行したものは，国鉄の座席予約装置におけるもの，および国鉄技研の Bendix G 15 に付加した装置では，いずれも磁気ドラムを遅延線の形で用いた．

ことに後者の場合は正規の G 15 の機能を一切乱さないことにしたため，command や word の構成，その他に大きな制約をうけたが，それでもなお，総合された演算速度は一挙に数十倍に向上させることができた．座席予約装置では少ない部品で相当能率のよい動作が実行されている．

2. 装置の構成およびデータ命令の形式

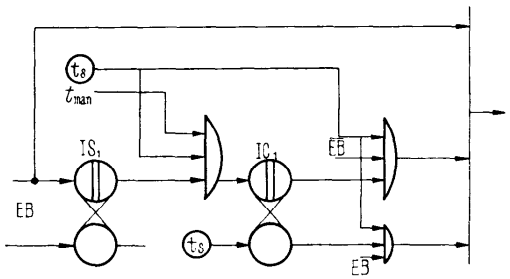
座席予約装置における優先順位をつけた．要求座席パターンのサーチング，ファイル訂正などのブロック処理方式については，すでに発表してあるので触れないで，ここでは一般の general purpose の演算について説明する．



第 1 図 全 体 の 構 成

第 1 図に全体の模型的構成を示す．一般のデータは長い遅延線 L, L′, L″……などの中にそれぞれ n 語ずつ入って循環している．A は 1 語の，B は 2 語の，C は $n/2$ 語の遅延線である．

線 N は各語の番号を示すデータが循環している．Γ は 1 語長の遅延線で command register の動的な部分である．この他に flip flop からなる 1 語長の shift register E および F がある．さらに IG_0, IG_1, IG_2 の 3 個の Inverting Gate（以下 IG とかく）がある．これをデータが通過するとき指令によって，絶対値と符号であらわされている数を，正数はそのまま，負数は 2′ s complement にしたり，その逆を行ったり，符号を変えたり絶対値だけにしたり，また数が 0 のあることを検出したりする．そのほか floating point の数の，exponent 部と mantissa 部を分けることもする．この基本的な図を**第 2 図**に示す．



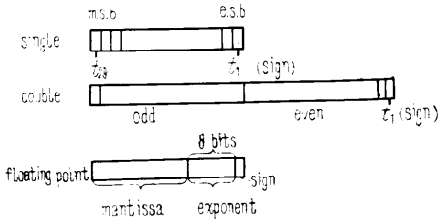
第 2 図 IG の 一 例

この IG はそれぞれ，遅延線より読み出したデータが入る Early Bus: EB_0, EB_1, EB_2 に属している．IG は数の流れている途中で，その性質を検出して，必要な変換を加えるために用いられる回路で，流れを遅らせることはない．ここで IG に用いられている flip flop は，ここに直接データが流れこまないときには，他の目的の 1 bit storage に用いられる．さらに遅延線に書き込まれるデータは Late Bus: LB を通って Destination の遅延線に書き込まれることが多い．これらの遅延線を EB, LB, IG および Adder 回路などをはさんで結合し，データを移動させることによってデータ群の処理を行うのである．これらの線は，命令レジスタの静的な部分，すなわち flip flop の出力によって，必要な道が作られ，不必要の道は閉されて結合される．この作られた道を通してどれだけのデータを移動させるかは，命令レジスタの動的な部分より発生する時間信号によって決定される．

また，第 1 図の flip flop: GF および E, F regis-

ter の最下位の flip flop: E_1, F_1 は，処理の途中における情報や符号などの一時記憶に用いる，flip flop: IP は2語 register : B の sign に関係する．

データは2進 29 bits で一語を作り，double length は 58 bits で表し，それは奇偶の address を占める．最右端の bit は符号である．そのあらわれる時間を t_s で表わす．(**第3図**参照)



第 3 図　語 の 構 成

また floating point の表し方は，始めの 20 bits が mantissa，次の 8 bits が exponent (excess 2^7) 最後が符号である (double length ももちろん考えられるが，ここでは説明省略)．これらの数は最下位の符号を先頭にして，遅延線から serial に読み出されてくる．1語の時間を $1wt$ とかき，word の中の各 bit に対応する bit time を考えると，1語は 29 bits でそれは t_1, $t_2 \cdots t_{29}$ のパルスよりできている．mantissa 部のあらわれる時間を t_{man}, exponent を t_{ex} などで区別する．これらの信号は counter や line N などより作られる．

命令形式はいろいろ考えられるが，つぎのことが指定される必要がある．

(a)　source および destination の遅延線番号，

(b)　ブロックデータの初めの address とその長さ，

(c)　OP code およびつぎの命令のよみとる address (時間)，

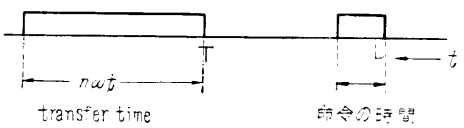
である．

OP code の中には，データが single か double か，関係する IG の動作などを盛り込めばよく．命令が1語の長さを越えるものがあっても，命令自身の double length 記号をつけておけば差支えない．

前に述べたように，命令の静的部分は，command register の静的部分に入り，必要な gate の開閉を準備する．line の中のデータ群の address を指定する部分は，最初のデータ位置 T とデータの個数 n であるが，まず T, n 否定がとられ，$\overline{\mathrm{T}}$にはその命令のある場所のアドレス T_0 が N line より読み出されTに加わって，$\overline{\mathrm{T}}+\mathrm{T}_0$ および $\bar{n}$ という形で，1語の遅延線 Γ にかき込まれ，1 word time ごとに $\overline{\mathrm{T}}+\mathrm{L}$ に1が加わると，指定した位置 T の前の word time で overflow pulse が生ずる．

これが出た次の wt の t_1 から，すでに準備されている path にデータを流す．このときから $\bar{n}$ に $1wt$ ごとに1を加えて，それが overflow する wt の t_{29} で流れを止める．この時間関係は**第4図**に示す．

この流れを transfer という．



第 4 図　命令の読み出しと実行時間との関係

3. ブロック演算の種類

ブロック演算をどの程度まで hardware の機能として取りつけておくかは，性能対費用の割合で定められる問題であるが，筆者の経験と第1図の構成から無理なく定まり，しかも必要度の多いものをブロック演算ができるようにしたまでであって，ここに示すものよりの拡張，あるいは縮少はもちろん可能なことである．次に考えたブロック演算の種類をあげる．

ここで L_i は一つの遅延線における i 番目（番地）の語を表す．

L_i' は同じまたは他の遅延線の対応する語を示す．

i は遅延線の循環周期が p word time であれば

i=remainder of i/p

を意味している．

(I)　copy

(a)　単純な copy: $\mathrm{L}_i \rightarrow \mathrm{L}'_i$

(b)　遅延 copy: $\mathrm{L}_i \rightarrow \mathrm{L}'_{i+1}$, $\mathrm{L}_{i+1} \rightarrow \mathrm{L}'_{i+2}$ etc.

(c)　符号変換する copy: $\mathrm{L}_i \times (-1) \rightarrow \mathrm{L}'_i$

(d)　絶体値の copy: $|\mathrm{L}_i| \rightarrow \mathrm{L}'_i$

(e)　負の数値を消去: $\mathrm{L}_i \rightarrow \mathrm{L}'_i$, 但し L_i の内容が負であれば0にする．

以上の (a)～(e) の操作を single および double length の word 群に対して動作させる．

(II)　summation

$\sum_i \mathrm{L}_i \rightarrow E$, $\sum_i \mathrm{L}_{i,\ i+1} \rightarrow B$

(III)　vector addition

$\mathrm{L}_i \pm \mathrm{L}'_i \rightarrow \mathrm{L}''_{i+1}$; $\mathrm{L}_{i,\ i-1} \pm \mathrm{L}'_{i,\ i+1} \rightarrow \mathrm{L}''_{i+2,\ i+3}$ (i=even)

(IV)　multiplication

(a)　$\mathrm{L}_i \times \mathrm{L}'_i \rightarrow \mathrm{L}''_{i+1,\ i+2}$ (積は double

length) (i=odd)

(b) $L_i \times L'_i \rightarrow L''_{i+2}$ (積は single length)

(c) $\sum L_i \times L'_i \rightarrow B$ (inner product, double len-gth)

(d) $A \times B \rightarrow B$, $A^n \times B \rightarrow B$ (double)

(e) $AB \rightarrow L_i$, $A^2B \rightarrow L_{i+2}$, $A^3B \rightarrow L_{i+4}$, $A^4B \rightarrow L_{i+6} \cdots\cdots A^nB \rightarrow L_{i+2(n-1)}$ (single) (i=odd)

(V) floating point addition and summation

(a) $A \pm E \rightarrow E$

(b) $\sum_i L_{4i} \rightarrow E$

(VI) floating point multiplication

(a) $L_{4i} \times L'_{4i} \rightarrow L''_{4(i+1)}$

(b) $A \times B \rightarrow B$, $A^n \times B \rightarrow B$

(c) $AB \rightarrow L_i$, $A^2B \rightarrow L_{i+4}, \cdots\cdots A^nB \rightarrow L_{i+4(n-1)}$

(VII) Selection

(a) max $|L_i| = L_k \rightarrow E$; k of $L_k \rightarrow A$

(b) first non zero $L_i = L_k \rightarrow E$; k of $L_k \rightarrow A$, set L_k=O in L

(VIII) Miscellaneous

(a) $L_i, {}_{i+1} \cdot B \rightarrow L_i, {}_{i+1}$ (extract)

(b) $L_i \times 2 \rightarrow L_i$ (left shift)

(c) $L_i \times 1/2 \rightarrow L_{i+1}$ (right shift)

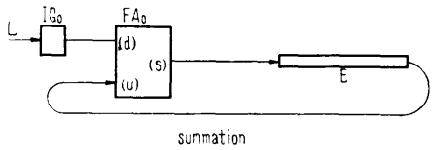
(d) $L_i, {}_{i+1} \rightarrow L_{i+1}$ (double→single)

4. 各演算の流れの説明

(I) の copy operation においては，source line より，early bus EB_0 に入り IG_0 を通り，late bus: LB に行き，destination line に至る道が開かれる．transfer の時間には destination line の循環は止まる．単純な copy は transfer の時間にデータが移動するだけであるが，途中に A, B などの delay line を経由すると番号 i がずれる．初め，A あるいは B にあったものが L に入り，最後に読み出された word が A または B に残る．また異った循環周期の line に入ると，i の位置が一循環のたびにずれて行くから，データをブロックで別の位置に移動できる．

(c), (d), (e)の操作は，データが IG_1 を通過するとき，sign bit を反転したり，落したり，検出して負であればその word を通過させないなどの単純な gate 動作である．

(II) summation は source line より Bus 1 に入り，IG_1 を通過してデータは加算形（負数は sign bit 1 と 2's complement）に変わり，full adder No. zero (FA_0) の augend (u) に入る．一方すでに加算形になっている E にあるデータは左端の E_1 から，FA_0 の addend (d) に入り，(s) より和が作られて，それは E_{29} に入って E を右に移動していく(**第5図**)．



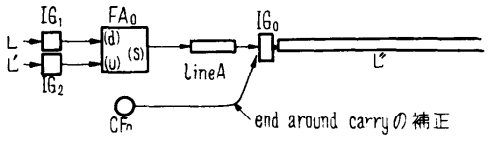
第 5 図 summation

その ward time の終りで end around carry がでて，次の wt の t_s (sign の bit time) で E_1 にある符号が訂正される．

transfer がつづけば，それはまた引続き FA_0 の (d) に入っていく．このようにして summation が作られていく．overflow の条件が生ずれば overflow indicator を set する．

double length の場合は E の代りに line B が用いられる．

(III) vector の加算では，line L_i は Bus 1 より IG_1 に入り加算形になる．line L'_i は Bus 2 に入り IG_2 を通り加算形になる．それらは FA_0 の (d) および (u) に入り，(s) より出たデータは line A に入り，1 wt おくれて sign bit があらわれたとき，FA_0 の end around carry で修正をうけ，early bus 0 に入り IG_0 を通って，絶対値と符号の形となって late bus に行き，destination line に入る (**第6図**)．

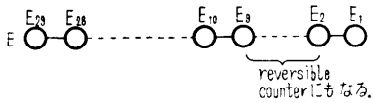


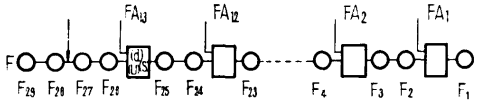
第 6 図 vector addition

double length のときは line A の代りに line B が用いられる．IG_1, IG_2 と FA_0 では over flow の検出は行われて，もし vector の要素が overflow すれば overflow indicator を set する．

(IV) 乗算の動作を説明するにはレジスタ E および F の構造に触れなければならない．これは他の line と異なって，それぞれ 29 bits の flip flop よりなり，$E_{29}, E_{28}, \cdots\cdots E_1$, および $F_{29}, F_{28}, \cdots\cdots F_1$, となっている．F レジスタは 2 bits ごとに full adder をはさんで途中から加算されるようになっている．この個数は FA_1 より FA_{13} までの 13 個である (**第7図**)．

E および F は右へ shift できるのであるが，特に E_{29}～E_{10} は左右どちらにも shift できる．



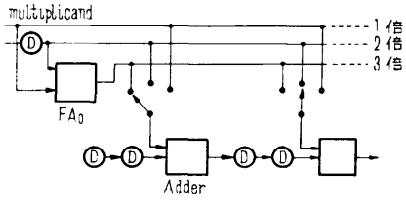


第 7 図　E および F register

E_9～E_2 は reversible counter の役目をする．F_9～F_2 は F_{29}～F_{10} と独立に shift できる．E_1, F_1 は sign を貯えるほかに 1 bit storage の役も兼ねている．E, F の多くの機能は floating point operation で必要となるものである．

乗算の時間は 2 *wt* を必要とする．これを ϕ_1, ϕ_2, とかく．ϕ_1 では，L_i は EB_1, IG_1 を通り，$|L_i|$ となって E レジスタに入っていく．L'_i は EB_2, IG_2 を通って $|L_i'|$ となって line A に入り，1 *wt* おくれる．積の符号は t_1 のときに，E_1 に set される．ϕ_1 の終りには E_{29}～E_2 に multiplier はおさまり，E_1 には積の sign がある．

ϕ_2 になると，$|L'_i|$ は line A よりあらわれてくるが，これは multiplicand であり，1 bit delay と full adder FA_0 を用い，multiplicand の 1 倍，2 倍，3 倍を作る．E レジスタの左から 2 bits ずつで multiplicand の 0, 1, 2, 3 倍の数を選んで，F レジスタの途中から加えていく．これは 4 進数の partial product を作って加えていることになる（**第 8 図**）．



第 8 図　multiplication の操作

$\phi_2 \cdot t_1$ の時に E_1 にある積の符号を late bus : LB に送り出し，$\phi_2 \cdot t_1$ から，F_2 から出てくる積を late bus に流すと，つぎの ϕ_1 にかけて double length の積が得られ，LB を destination line L″ に結合することによって（IV・a）の場合となり，ϕ_2 で積を late bus に入れず，つぎの $\phi_1 \cdot t_1$ で E_1 にある符号を，また $\phi_1 \cdot t_2$ の時間から積を LB に入れれば single の積が L″ に入る．double length のとき，late bus を EB_0 に結合し，IG_0 を通って積の summation を line B に作れば（IV・c）が求まる．

multiplicand を始めに line A におき，multiplier を line B において（符号は IP flip flop に入る）積を B に double length で作るのは（d）の始めの場合で，このとき積の符号は flip flop IP に作られる．これを他に移動する場合，double length でも，上半分あるいは，下半分の single でも IP が正しい符号の時間 t_s で送り出される．（d）の始めの動作をつづければ，2 *wt* ごとに line B の積に line A の数がかけられて $A^n \times B$ が line B の中に作られる．符号は IP にある．AB, A^2B, A^3B……が作られるときこれを single の形にして late bus を通して destination に入れていけば（IV・e）が作られる．

（V）floating point の加減算や summation は，4 *wt* が基準となる．これを ϕ_1, ϕ_2, ϕ_3, ϕ_4 とする．数が IG を通過するとき，t_{man}, t_{ex} などの時間のマスクが作用して，mantissa 部のみ変化を受ける．一つの数の mantissa 部を x, exponent 部を ξ で，他の数を同じく y, η で表す．

基本になる（V・a）の動作を説明すると，E レジスタの中にはすでに x, ξ が入っており，mantissa 部は加算形になっているとする．line A に (y, η) がある．

ϕ_1 では A は EB_1, IG_1 を通り，y は加算形に変る．t_{ex} の時間で η より $\bar{\eta}$ が作られ，full adder FA_0 の（u）に入り，E_9～E_2 にあった ξ は FA_0 の（d）に入る．また ξ は同時に F_9～F_2 に移されていく．FA_0 の（s）より $\xi+\bar{\eta}$ があらわれ E_9～E_2 に移される．$\phi_1 \cdot t_{10}$ の時間における FA_0 の carry flip flop CF_0 の状態は exponent の大小を示している．

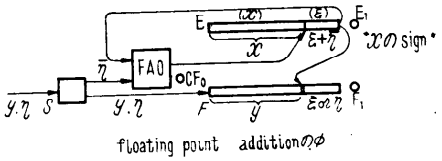
$CF_0=1$ ならば $\xi > \eta$, E_9～$E_2=\xi-\eta-1$

$CF_0=0$ ならば $\xi \leqq \eta$, E_9～$E_2=\overline{\eta-\xi}$

IG_1 よりあらわれた (y, ξ) はそのまま F_{29} より F レジスタに入っていく．sign は t_1 のとき，F_1 に set される．CF_0 の状態は t_{10} より保持されて t_{21} になったとき，F_{10} まで進んできた η を F_9～F_2 に流すか止めるかを定める．$CF_0=0$ であれば ξ の代りに η がおきかわる．したがって ϕ_1 の終りにはつぎのようになっている（**第 9 図**）．

E_{29}～$E_{10}=x$, E_9～$E_2=\xi+\bar{\eta}$, E_1=sign of x, F_{29}～$F_{10}=y$, F_9～$F_2=\xi$ あるいは η の大きい方, F_1=sign of y

ϕ_2 では，exponent の小さい方を shift して桁を



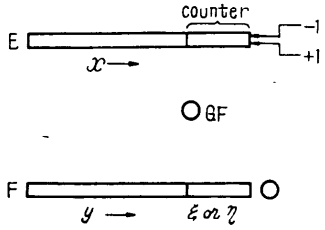
floating point addition の ϕ

第 9 図 floating point addition の ϕ_1

そろえる．x, y はすでに加算形になっているため，shift のとき左端より補充する bit は E_1, F_1 にある sign に支配される．この shift の数をきめるものは E_9～E_2 の数であり，ϕ_2 ではこれが reversible counter となる．

full adder FA_0 の CF_0 が 0 であれば E_{29}～E_{10}, $CA_0=1$ であれば，F_{29}～F_{10} が right shift する．$CF_0=0$ のときは，E_9～E_2 は $\overline{\eta-\xi}$ であるから，この reversible counter に 1 を $\eta-\xi+1$ bit time 加えれば counter は overflow pulse を出す．shift を control する flip flop は GF であり，GF を bit time 1 より $\eta-\xi+1$ bit time まで on にすることで，x は $\eta-\xi$ bit だけ right shift する．21 bit time になっても GF が off にならなければ，あとで $x=0$ にする．同様に $CF=1$ のときは counter には $\xi-\eta-1$ がおかれているから -1 を $\xi-\eta$ bit time 加えると，counter は under flow pulse を出す．

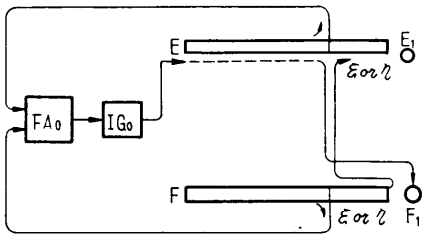
bit time 0 より $\xi-\eta$ bit time まで $GF=1$ になり F_{29}～F_{10} は right shift する．20 bit time 以上 GF が 1 であれば y を 0 にしておく．ϕ_2 の終りには加減算ができるように桁がずらされ，その exponent は F_9～F_2 にある（**第 10 図**）．



第 10 図 ϕ_2

ϕ_3 では桁がそろえられた mantissa の加減算が full adder FA_0 によって行われ，FA_0 の (s) より出た和は EB_0, IG_0 を通過して，和が $1-2^{-m}=11111\cdots\cdots000$ の 2's complement の形であるかどうか検出される．もし，この形であり，和の符号が負であれば，あとの normalize のときに補正が必要となる．

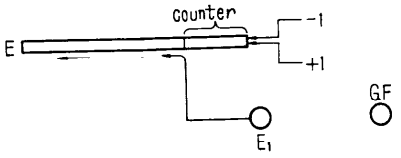
加算は ϕ_3 の t_1 より t_{21} にわたって行われ，結果は E_{29}～E_{10}, F_1 の 21 bits で表わされ，sign は E_1 に set される．t_1～t_8 の時間に F_9～F_2 にあった exponent は E_9～E_2 に移動する（**第 11 図**）．



第 11 図 ϕ_3

ϕ_4 では結果の normalize である．負の数は 2's complement で表わされているため，normalize は注意を要する．

正のときは E_{29} に 1 が表われるまで，負のときは 0 が表われるまで left shift をするが，すでに検出された $1-2^{-m}$ の 2's complement の形の場合は E_{29} が 0 になるまで left shift する．また left shift による exponent の補正は E_9～E_2 が再び reversible counter の役目をして行われる．left shift の bit time を k とすると $(k-1)$ bit time だけ counter を動作させ，-1 を加え exponent を補正する．$k=0$ のときは 1 を加える．この時間をきめるものは shift の bit time をきめる flip flop GF の on の時間である．exponent が overflow すればその indicator を on にし，underflow すれば結果をすべて 0 にしてしまう．ϕ_4 の終りには E_{29}～E_{10} には mantissa, E_9～E_2 には exponent, E_1 には sign がある（**第12図**）．



第 12 図 ϕ_4

このため，直ちにつぎの floating point の数が EB_1 にあらわれても新しいサイクルで加算が行われる．

データの source が line A の代りに L_{4i} であり，transfer の時間がつづけば V(b) が成立する．

(VI) floating point multiplication も 4 *wt* かかり，ϕ_1, ϕ_2, ϕ_3, ϕ_4 とする．(a) をまず説明する．

ϕ_1 では（IV·a）と同じように，L_{4i} は EB_1, IG_1 を通って mantissa の絶対値が作られ，かつ，それが 0 であるかどうか検出されてEレジスタに入る．L'_{4i} も同様に EB_2, IG_2 を通り，mantissa は絶対値がとられ，0 かどうか検出される．mantissa 部は line A に入り 1 *wt* おくれる．

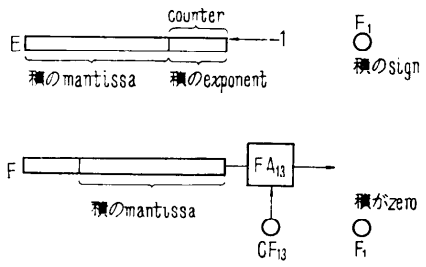
積の sign は E_1 に set され，積が 0 であることは F_1 に set される．ϕ_1 の exponent の時間 t_{ex} では両方の exponent ξ, η は full adder FA_0 に入り，和は E_9～E_2 に入る．$\phi_2 \cdot t_1$ で exponent は excess 2^7 に修正される．

積の exponent が 2^8 になれば overflow indicator をつけ，0になれば F_1 を set して結果を 0 にする．$\phi_2 \cdot t_2$ になると A から multiplicand があらわれ始めるから，fix point と同様に，その 0 倍，1 倍，2倍，3倍を作り，E_{29}～E_{10} にある multiplier で select しながら，F レジスタの途中に入れていくと 40 bits の積が作られる．$\phi_3 \cdot t_7$ の時間にはこのうち高位の 22 bits が F_{23}～F_2 に残り，ここで F の right shift を止める．$\phi_3 \cdot t_8$ で最高位の F_{23} が 1 であればFを 1 bit right shift, 0 であればそのままにしておき，E_0～E_2 は reversible counter となり，exponent に −1 を加える．これは shift を control する GF の on, off できまる．

これで mantissa は normalize されて F_{22}～F_2 にあり，exponent は E_9～E_2, sign は E_1 にある．また F の最下位にある full adder FA_1 の carry f.f を $\phi_3 \cdot t_8$ で1に set して，$\phi_3 \cdot t_9$ でFを 1 bit right shift し，mantissa 21 bit 目を0拾1入しておく．

ϕ_4 において，F_1 が set されていなければ $\phi_1 \cdot t_1$ で E_1 にある sign を，$\phi_4 \cdot t_{ex}$ で E_2 から exponent を，また $\phi_4 \cdot t_{man}$ で FA_1 の (s) から mantissa を LB に送り，destination line に入れる（**第13図**）．(VI·b) (VI·c) は L, L' の代りに A, B line を用いる．

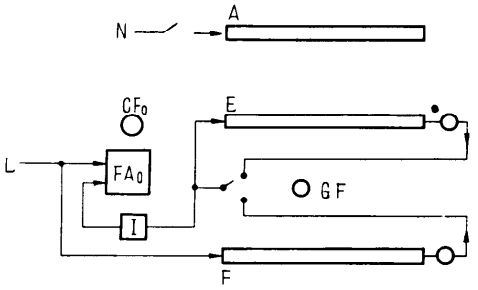
$A^n \times B$ は $4n$ *wt* ごとにできるのがfixed point と



第 13 図　floating point の multiplication

の相違である．

(VII) の selection の動作は L_i の絶対値の大きいものが常に E レジスタに残るようにデータが流れる（**第14図**）．



第 14 図　最大値の選択

L_i は EB_2, IG_2 を通り $|L_i|$ となり，full adder FA_0 の (u) に入ると同時に F_{29} より F レジスタに移っていく．Eレジスタの内容は shift して否定されて，FA_0 の (d) に入る．

このとき，flip flop GF=0 であれば，すでにEレジスタにあった内容は E_1 より E_{29} を経て E レジスタに循環する．t_{29} で FA_0 の carry flip flop CF_0 が1に set されれば，新しく L_i からきたデータの方が E のデータより大であるから，GF=1 にしてつぎの *wt* ではFレジスタを E レジスタに shift すると同時にその否定を FA_0 の (d) に入れ，その時に新しく入ってくるデータと比較する．GF=1 になった word time に number line N の word を line A に移すことによって，それまでの最大の数の address は lineA に残り，その数自身は E に残ることになる．GF=0 であれば，新しいデータは E に残っているいままでの最大値と比較されることになる．

(VII·b) では L_i は EB_2, IG_2 を経てEレジスタに移る．IG_2 では non zero detection が行われ，non zero の数が現れたらその word time の t_{29} で GF が set される．

GF=1 は 1 *wt* つづき，このとき N line の数は Aに移り，Lの循環を止め，GF·t_{29} で transfer を止める．

したがって，最初の non zero の数は E レジスタに残り，その address は A にあり，L の中のその non zero の数は0になる．再びこの命令を用いれば，次の non zero の数とその address がわかる．

(VIII·a) はLと Bをそれぞれ EB_1, EB_2 に入れ，そ

の and をとって late bus に入れて destination line に入れる.

(Ⅷ·b) は循環の途中に 1 bit delay を入れる. sign bit は delay を通さず, t_2 では 0 を書き込む. これで sign を除いて left shift する.

(Ⅷ·c) では L と E とを結合し, データは E_2 からもとにもどす. sign は E_1 を経由させる. t_{29} では L に zero を補給する. これで sign を除いて right shift する. double length のときは L と A と E との結合になり, 動作は前と同じである.

(Ⅷ·d) では L と A を結合し, sign だけ A の右端からもどし, L の右端は single の絶対値だけを L の左端にもどして循環させる. これで double length は single length になり, sign は正しい位置におかれている.

以上で 3 節に述べたブロック演算のデータの流れの大体の説明である. 計算機としてはこの他の命令, たえとば割算, 条件付飛越し, 入出力など種々あるが, ブロック演算には関係がないので省略する.

5. プログラムの例

ブロック演算は演算速度の向上だけでなく, プログラミングの簡易化ともなることを二, 三の例で示す.

(1) a_i, b_i (i=1～100) が与えられて $\sum_1^{100} a_i b_i$ を計算する. a_i b_i は別々の line の対応する位置にあるとする. (同じ位置になければ block copy で同じ位置にすぐ移る) long line には 128 words あるとする. 命令の形式は次のようになる.

1. clear line B
2. make $\sum a_i b_i$ (i=odd)→B
3. shift one word a_i
4. shift one word b_i
5. make $\sum a_i b_i$ (i=even)→B

line B には $\sum a_i b_i$ が作られる. 最大 5 circulation time 以内で演算は終了, 始めからデータの配列を考えておけば $\sum a_i b_i$ は 1 circulation 以内で作ることができる.

(2) $\sum a_i x^i$ を作る. a_i は相つづく odd address にあるとする.

1. x→line A
2. 1→line B
3. make $x \cdots\cdots x^n$ in line L
4. clear line B
5. make $\sum a_i x^i$ in line B

これは 2 circulation 以内で終了する.

(3) $a_1 \cdots\cdots a_{100}$ までを大きさの順序にならべかえる.

1. select max. value in E and mark its address in A.
2. clear location indicated by A register. (max の data のあつた位置が clear される)
3. precess line L′ through E. (L′ に E の内容が line L′ の右端から入っていく).
4. decrement counter.
5. jump to 1 if counter is not zero.
6. stop. (line L′ に a_i が大きさの順序にならぶ)

大体 300 circulation time 以内に sorting は終了する. このように高速化とプログラムの簡易化が同時に達成できた. この例のようなブロック演算に適した問題では, 従来のドラムを用いた機械の 100 倍以上の速度となる. 以上のべた方式の大部分を Bendix G 15 にとりつけたら floating の vector を取扱う問題において従来用いていた Bendix G 15 の 1/60 の時間で計算が終了したと報告されている[3].

6. あとがき

遅延線を用いるとブロック演算が容易にできることに気付き, その考を special purpose computer では座席予約装置に, general purpose computer としては Bendix G 15 の付加装置に応用した. 遅延線はいずれも磁気ドラムであるが, 演算要素は前者では transistor と diode logic を用い, 後者では Bendix 本体との結合の問題もあって, Bendix と全く同じ真空管と Ge diode logic を用い, いずれも十分安定に動作している. これらはすでに 2 年以上前に設計は終り, 機械の方はいずれも 1 年近く動作をしている. しかし, 始めから general purpose computer を制約なく設計し, 遅延線に 1 Mc～2 Mc 位の clock の magnetostrictive なものを用いれば, 小形で僅かな材料で高速な計算機が作れるであろうと思っている.

終りにこの種の機械を設計するにあたって, 種々討議して頂いた鉄道技研自動制御研究室の諸氏ならびに製作に当たられた日立戸塚工場の方々, 三菱電機研究所, 無線機製作所の方々に厚く御礼申しあげる.

参考文献

1) 穂坂: 電通学会 33 年秋期大会シンポジウム予稿
2) 穂坂, 大野, 谷: 電通学会 34 年秋期大会シンポジウム予稿
穂坂, 大野: 予約機械, 共立出版, 34年
3) 鈴木: 数理科学綜合研究第 4 班第 1 回プログラムシンポジウム, 35年 1 月