On Block Operations Using Delay Lines

MAMORU HOSAKA*

1. *Introduction*

In designing a small scale computer, serial operation, serial memory, and slow clock rate are often selected, because cost is limited and stable operation with easy maintenance is most desirable. Consequently its large access time and low operation speed make it difficult to get good point in performance vs. cost ratio, compared to large scale computers with random access memory and parallel high speed operations. However, small scale computers are useful in those area where problems to be solved are not so large and not complicated, that solutions can be obtained within reasonable time.

Usually one of objectives for use of automatic computers is to solve problems which are not small and occur frequently in research, design and data processing works. As most of those problems fall in limited number of categories, small scale computers provided with special devices for only such categories can be effectively used. Thus problems so for given up to try with them will become to be solved. For example, those which include large scale vector or matrix computations, sorting or frequent table lookup operations, are the cases, in which there need great quantities of data manipulations and iterations of operations. In stored program computers, iteration techniques have to be used as much as possible to make programs short, but if address modification, counting and other book-keeping operations are too many, comparing to amounts of pure data monipulation, overall processing time will be longer. On the contrary if the rate of bookkeeping operations are small, processing time will not increase without build-in index registers and other bookkeeping facilities.

For address modification even with build-in mechanisms, it is necessary to arrange data in regular order by some means. To process one of these data, a command is read from memory, modified, decoded, executed, and data is read from memory, manipulated, number of times counted, its limit detected, and program branched. If data manipulation is simple, these red tape operations make processing time longer and programing cumbersome. If one build-in command can work on all the specified data regularly addressed, without using subroutines, high speed processing as well as easy programming can be attained. These operations which work

* Aeronautical Research Institute, Univ. of Tokyo

on a group of data are called block operations.

In order to do efficient block operations, data are to be read, processed and stored back to memory without data flow interruption. For this purpose, delay line type memories are most suited, in which data are circulating with regeneration. By connecting delay lines with logical circuits in between for specified flow time, a group of data are transferred, being processed automatically.

In recent developed large scale computers, bookkeeping or red tape operations are designed to be executed concurrent with main processing routines. Block operations stated here is not executed in such a way, but a group of data located in contiguous positions are processed faster by one command, without using too much hardwares.

Any type of delay lines can be used in block operations, but so far the author used those of magnetic drum type successfully in the Seat Reservation System of Japan National Railways and in the speed up device attached to the Bendix G-15 computer at the Railway Reserch Laboratory of Japan National Railways. In the former, block operations are considered in searching with priority for required seat position pattern and updating of the seat file, whose reports were already published[1]. In the latter, many restrictions were imposed in order to connect to the existing computer, but its speed for group data manipulations raised as high as hundreds times. In this paper the design of the latter case is described.

2. *Structure of the Device and the Formats of Data and Commands.*

Main parts of the device consist of delay lines and registers as shown in Fig. 1. Delay lines L's are of n word length, line A one word length and line B two word length. Line N contains n words in each of which there is numbers showing the word time or word address of the corresponding location in the line L's. Line Γ is used for determining the word time duration in which block operation continues. Contents of the line Γ is supplied from the address part of a command. And there are two one-word registers E and F which are made of flip-flops, constituting shift registers. IG_0, IG_1 and IG_2 are inverting gates whose function are to convert numbers i.e. from the normal form (sign plus absolute value) to the addition form (sign plus 2's complement, if negative) or vise versa, to change sign, to take absolute value, to detect zero, to separate exponent and mantissa of floating point numbers, and when IG's are not used these ways, their flip-flops are used for one bit storages. The schematic circuit of IG's is shown in Fig. 2. By commands, data read from one of delay lines are transferred to one of destination lines or registers, via early bus EB_i, inverting gate IG_i ($i=0, 1, 2$), appropriate

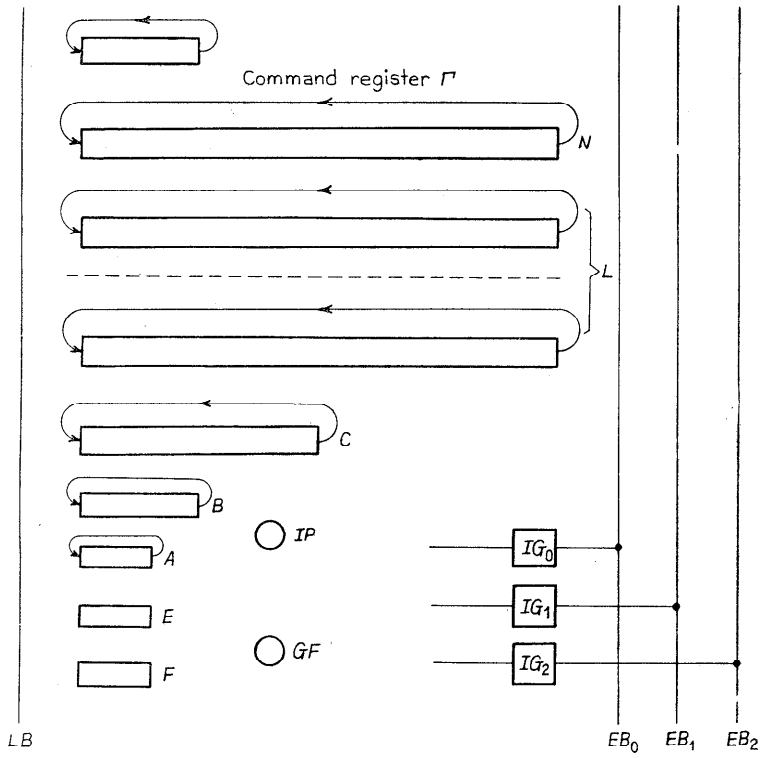


FIG. 1. Schematic diagram

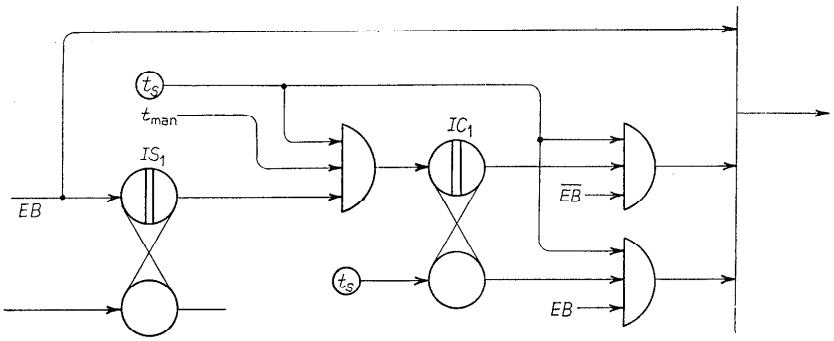


FIG. 2. Example of inverting gate

logical circuits and late bus *LB*. During transfer process, data are manipulated by gates which are controled by a command. The function of command is to set the data flow paths and to determine the time interval to flow the data in the selected paths. E_1 and F_1 are usually sign bit flip-flops of E and F registers, the flig-flop IP indicates sign of

product in the line B. The flip-flop GF is primarily used for the control of shift in E and F register. Otherwise these flip-flops are utilized for one bit storages.

Numerical data of single precision are expressed by one word of 29 bit length, double precision is of 58 bit length which occupies two consecutive even and odd addresses. The right most bit of numerical data indicates sign which is read from a delay line at first bit time t_s. In a floating poit number, left 20 bits are used for mantissa, next 8 bits are for exponent (excess 2^7 expression) last one bit indicates sign (Fig. 3).

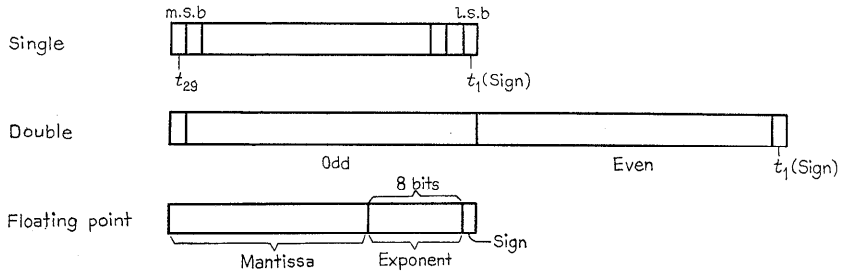


FIG. 3. Stracture of word

These data circulate serially in delay lines, flowing with sign bit first and most significent bit last. The time interval of one word length is indicated by a word time which consists of 29 bit time $t_{29}, t_{28}, \cdots, t_1$. In a single precision word, at the bit time t_1, sign bit is read, at t_{29} most significant bit read, in a double precision word, t_s (t_1 of even word time) is sign bit time. In a floating point number of single length, exponent emerges at bit times, $t_{\exp}$ ($t_2 \sim t_9$) and mantissa at bit times t_{man} ($t_{10} \sim t_{29}$). Timing signals are generated from the line N and the counters.

In block operation commands, at least next items have to be specified, except (d). (a) type of operation (OP code). (b) line numbers of data source and destination. (c) address of first data and word length of block data. (d) next command address. OP code is the static part of a command, which determines the data paths from a source to a destination line via a certain bus and logical circuits. The time interval for flowing data is determined by the information in item (c) as follows. If the word time when the first data to be processed come from a source line is T and the group data length is n words, $c_1 = T_0 + \overline{T} + 1$, $c_2 = \overline{n} + 1$ are written in the line Γ at the command read time T_0; $T_0 + 1$ is obtained from the output of the line N. Then, one is added into c_1 part in Γ each word time, generating overflow at the word time $T-1$, by which the transfer gates of the selected data paths are opened from the begihning of word time T. Just as the same way, one is added to c_2

part in the line Γ from the word time T, then the overflow pulse at the word time $T+n-1$ cause the transfer gates closed. The timing relation between the command read time and the data transfer times is shown in Fig. 4.

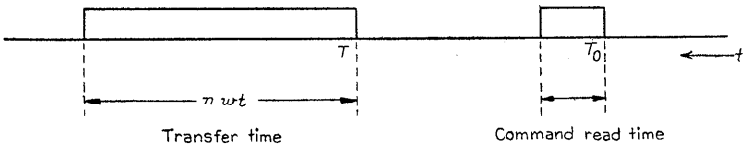


FIG. 4. Command read time and its execution time

3. *Types of Block Operation.*

To what extent block operations are build-in as hardwares is to be determined by performance to cost ratio, and frequency of their use. Considering this problem, the author constructed block operation circuits as shown next with not too much components used. Now L_i, L'_i etc. indicate a word located at address i of delay lines L and L' etc. If the recirculation period of a delay line is p word times, i is replaced by the remainder of i devided by p.

(I) Copy operations
- (a) Simple copy: $L_i \rightarrow L'_i$,
- (b) Delayed copy: $L_i \rightarrow L'_{i+1}$, $L_{i+1} \rightarrow L'_{i+2}$ etc.
- (c) Inverted copy: $(-1)\ L_i \rightarrow L'_i$
- (d) Absolute value copy: $|L_i| \rightarrow L'_i$
- (e) Negative number elimination copy: If L_i is negative value, copyed values are set to zero, otherwise simple copy.

These copy operations works on a word group of single or double precision.

(II) Summation
$$\sum_i L_i \rightarrow A\,, \quad \sum_i L_{i,i+1} \rightarrow B$$

(III) Vector addition or subtraction
$$L_i \pm L'_i \rightarrow L''_{i+1},\ L_{i,i-1} \pm L'_{i,i+1} \rightarrow L''_{i+2,i+3}$$

(IV) Multiplication
- (a) Block multiplication (double): $L_i \times L'_i \rightarrow L''_{i+1,i+2}$ $(i=\text{odd})$
- (b) Block multiplication (single): $L_i \times L'_i \rightarrow L''_{i+2}$
- (c) Inner product (double): $\sum_i L_i \times L'_i \rightarrow B$
- (d) $A^n \times B \rightarrow B$ (double)
- (e) $AB \rightarrow L_i$, $A^2B \rightarrow L_{i+2}$, $\cdots$, $A^nB \rightarrow L_{i+2(n-1)}$, single product $(i=\text{odd})$

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

(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 $L_k = 0$ 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. *Data Flow and Control in the Block Operations.*

(I) In the copy operation, the data path opens from the source line to the destination lines via the ealy bus EB_0, the inverting gate IG_0 and the late bus LB. In the transfer time, the recirculation of the destination line is stopped and the source data are copyed into it without delay. In the delayed copy, the data path includes the other delay line so that the word time of copyed data are delayed, i.e. suffix i increases. When the recirculation periods of the source and destination lines are different, copyed location numbers are shifted in each recirculation, accordingly the block shift operation is possible. In the operation, (I_c), (I_d), and (I_e), data are converted by the gates of IG_0 to the appropiate form.

(II) In the summation operation, data from L go to the augend entry of the full adder FA_0 via the early bus EB_0 and the inverting gate IG_0 where numbers are converted to adding form, that is, negative number is expressed in 2's complement and sign bit. While the partial sum in the line A goes to the addend entry of full adder FA_0 and the sum goes to the line A as shown in Fig. 5. If the end around carry occurs, the sign of the sum is corrected by the action of carry flip-flop CF_0 of FA_0

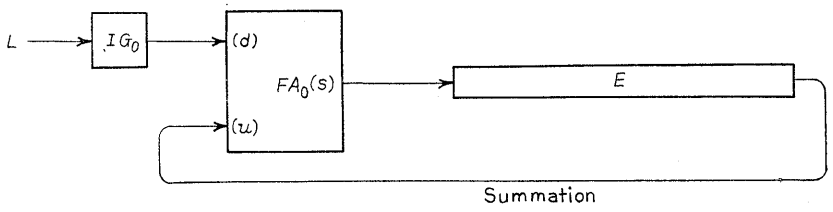


FIG. 5. Summation

at the bit time t_1 of the next word time. If the data transfer time continues, data from L go to FA_0, consequently their summation is produced in the line A. When overflow occurs, the overflow indicator is set. When double length words are summed, the line B is used insted of the line A.

(III) In the vector addition and subtraction, components of the two vectors are located in the word time positions of line L and L'. Data from line L and from line L' go to the full adder FA_0 via EB_1, IG_1, and EB_2, IG_2 respectively, while their component sum goes to the line L via the line A and IG_0 as shown in Fig. 6. At the exit of the line

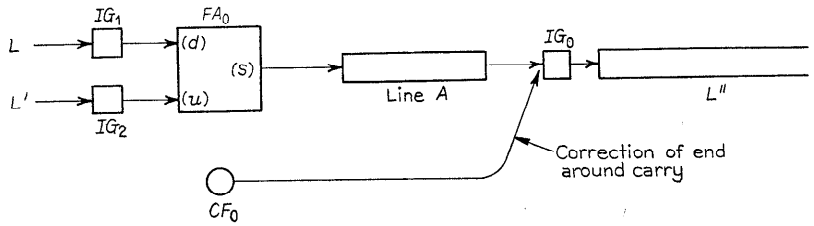
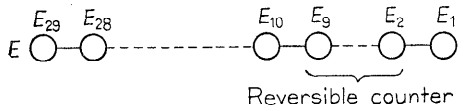


FIG. 6. Vector addition

A, one word time later from the out put of FA_0, the sign of the sum are corrected by the end around carry at FA_0. IG_0 is used for the conversion to the normal from (absolute value and sign). The overflow of the components sum is detected by IG_1, IG_2 and FA_0, setting the overflow indicator.

(IV) Before entering the multiplication operation, the structure of the registers E and F must be understood. These registers are made of the static flip-flops $E_{29}, E_{28}, \cdots, E_1$ and $F_{29}, F_{28}, \cdots, F_1$. In F register, full adders $FA_1, FA_2, \cdots, FA_{13}$ are inserted every two flip-flops as shown in Fig. 8, so that numbers can be added sierially at shifted positions into



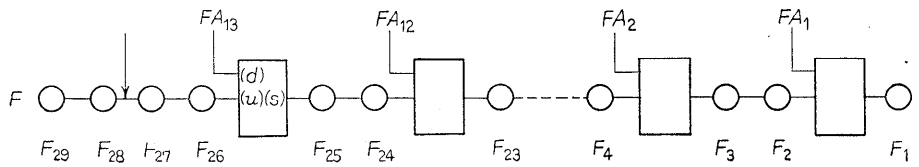


FIG. 7. E and F register

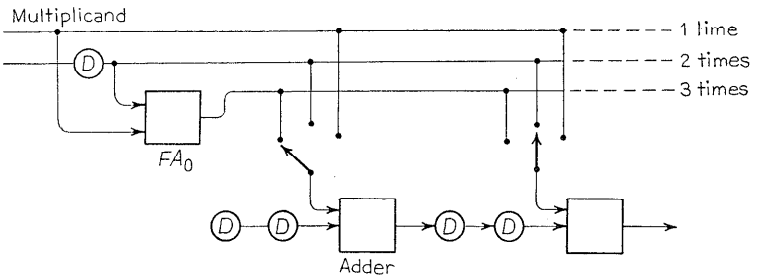


FIG. 8. Multiplication

the right shifting content of F register. Data in E and F can be shifted right. In $E_{29}\sim E_{16}$, left shift is also possible and $E_9\sim E_2$ are used frequently as the reversible counter. The least significant flip-flops E_1 and F_1 store sign bit, otherwise they are used for one bit stroges in the course of operations. Most functions of E and F registers are provided for the floating point arithemtic.

Multiplication takes two word times which are denoted by phase 1, ϕ_1, and phase 2, ϕ_2. In ϕ_1, the multiplier in L goes to the register E via EB_1, IG_1, where its absolute value is taken, while the absolute value of multiplicand from L enters into the line A, the sign of the product is set in E_1 at the bit time t_1. At the end of ϕ_1, the multiplier is in $E_{29}\sim E_2$ and the multiplicand in the line A.

In ϕ_2, the each quaternery digit (which are made of consecutive two binary digits, making fourteen quadriary digits.) in the register E, selects one of zero, one, two, and three times of the multiplicand according to the quaternery number zero, one, two, and three, and add it through the full adders FA_j ($j=1\sim13$), which corresponds to the quaternery digit position from the right, to the shifting F register (when $j=14$, F_{27} is used instead of FA_{14}). The multiple values of the multiplicand are made by the full adder FA_0 and one bit delay when it comes from the line A. The two word length product are thus produced at the time from $t_2\cdot\phi_2$ to the next $t_{29}\cdot\phi_1$ and sent to the destination line L with the product sign bit stored at E_1 attached at the bit time $t_1\cdot\phi_2$. When the products are sent to the line B via EB_0, IG_0 and the full adder FA_{14} connected to the line B, then the inner product is made in the line B just as the same process in (II). If the sign bit is sent at bit time $t_1\cdot\phi_1$ and the product is send to the line L from $t_2\cdot\phi_1$ to $t_{29}\cdot\phi_1$, the single precision products are obtained. These operations are the explanation of the cases (IVa) (IVb) and (IVc).

In (IVd), the multiplicand is placed in the line A, the multiplier in the line B and its sign in IP flip-flop. After the multiplication transfer time of two word length, by almost the same process stated above, the

product is made in the line B, its sign is in IP flip-flop, the line A recircurate unaltered, so that when the transfer time continues $2n$ word times, $A^n \times B$ is produced in the line B with the sign in IP. During this process $A^i \times B$'s can be copied into the line L in single length form, this is the case (IVe).

(V) In the floating point addition and subtraction, the operation is divided into the four phases ϕ_1, ϕ_2, ϕ_3, and ϕ_4, each of which takes one word time. A floating point number is expressed by (x, ξ), where x is its mantissa and ξ its exponent. Also (y, η) is another floating point number. When floating numbers pass through the inverting gates, only manrissa parts are subject to their control and exponent parts are extracted.

In the case (Va), an augend is in E register. its mantissa being in addition form (x', ξ) i.e. 2's complement if negative, and an addend is in the line A in the normal form (y, η). In phase one, ϕ_1, (y, η) comes from the line A to IG_1 via EB_1, where y is converted to the addition form y' and $\bar{\eta}$, separated from (y', η), goes to the augend terminal of full adder FA_0. At the same time ξ in $E_9 \sim E_2$ goes to its addend terminal, making their sum $\bar{\eta}+\xi$ which goes into $E_9 \sim E_2$ and ξ is transferred into $F_9 \sim F_2$. At the bit time t_{10} of ϕ_1 $(t_{10} \cdot \phi_1)$, the carry flip-flop CF_0 of FA_0 shows magnitude comparison of ξ and η, that is,

If $CF_0=1$: $\xi > \eta$, $E_9 \sim E_2 = \xi - \eta - 1$

If $CE_0=0$: $\xi \leqq \eta$, $E_9 \sim E_2 = \overline{\eta - \xi}$

The condition of CF_0 is hold to the end of this phase. (y', η) from IG_1 enters into F register, shifting from F_{29} to the right, its sign is set in F_1 at the bit time t_1. At the bit time t_{21} of ϕ_1, whether η_1, which is now in $F_{17} \sim F_{10}$, is allowed to proceed into $F_9 \sim F_2$, is determind by the condition of CF_0 so as to leave the greater exponent in $F_9 \sim F_2$. At the end of phase 1, ϕ_1, data in E and F registers are as follows,

$E_{29} \sim E_{10} = x'$, $E_9 \sim E_2 = \xi + \bar{\eta}$, E_1=sign of x

$F_{29} \sim F_{10} = y'$, $F_9 \sim F_2 = \xi$ or η, which is greater, F_1=sign of y

as shown in Fig. 9.

In the phase two, the mantissa with smaller exponent is shifted right

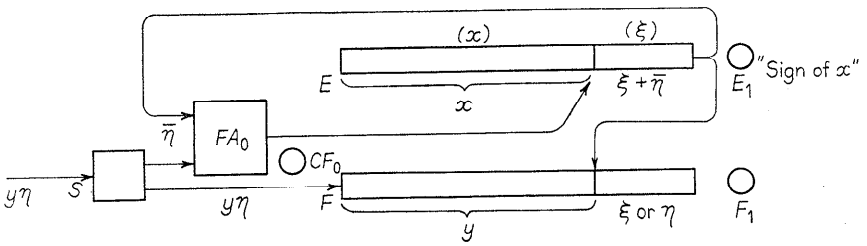


FIG. 9. ϕ_1 of floating point addition

to equalize exponents of the augend and the addend. Bits supplied from the left side in shift operation depend on the sign bit in E_1 or F_1 and the number of shifting bit times is determined by on-time of the flip-flop GF controlled by the reversible counter $E_9 \sim E_2$. When the carry flip-flop $CF_0=0$, the shift control flip-flop GF is set at the bit time $t_1 \cdot \phi_2$, and reset by the overflow pulse of the reversible counter, which counts $\eta-\xi$ bit shifts of $F_{29} \sim F_{10}$. When $CF_0=1$, GF is set at $t_{29} \cdot \phi_1$ and reset by the underflow pulse of the reversible counter which counts $\xi-\eta$ bit shifts of $E_{29} \sim E_{10}$. In both cases, if GF is on more than twenty bit times, the corresponding mantissa in E or F is made zero. At the end of the second phase, both mantissas are in $E_9 \sim E_2$, and $F_9 \sim F_2$, the exponent in $F_9 \sim F_2$, sign in E_1 and F_1 as shown in Fig. 10.

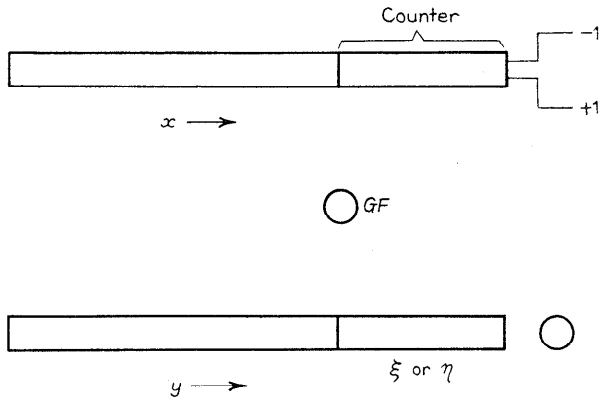


FIG. 10. ϕ_2

In the phase three ϕ_3, x' and y' with the same exponent are added by passing the full adder FA_0 and the sum of 21 bits goes to $E_{29} \sim E_{10}$, F_1 and its sign to E_1, via EB_0, IG_0, where the test is made if the sum is in 2's complement form of $1-2^{-m}=111\cdots100\cdots0$ with negative sign. If this is the case, the special normalizing is made at ϕ_4. In bit times $(t_1 \sim t_8) \cdot \phi_3$ the exponent in $F_9 \sim F_2$ is transferred to $E_9 \sim E_2$. Flow paths in ϕ_3 are shown in Fig. 11.

In the phase four, the result is to be normalized. The number in $E_{29} \sim E_{10}$ and F_1 is shifted each bit time to the left until one is set in E_{29} if $E_1=0$, or until zero is set in E_{29} in the special normalizing case, otherwise until 0 is entered in E_{29} if $E_1=1$. The special case ocurrs because the negative number is expressed in 2's complement form when normalizing.

The correction of the exponent with the left of the mantissa is made by the reversible counter which counts the left shift bit times minus

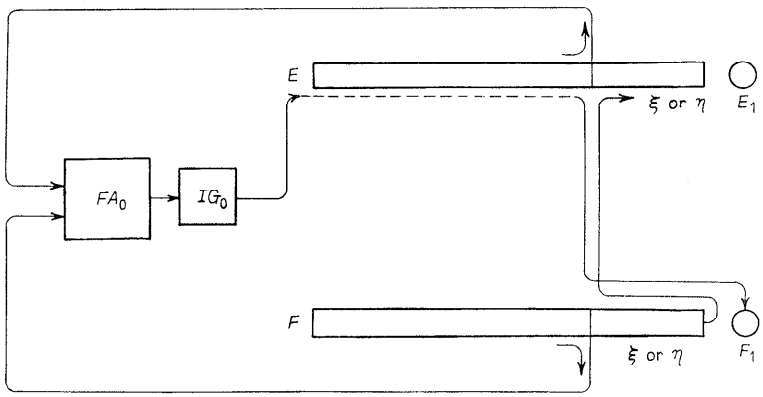


FIG. 11. ϕ_3

one. When no shift ocurrs, the exponent increses one. The left shift bit time is directly controlled by the on time of GF just as the right shift case in ϕ_2. When the exponent overflows, the overflow indicator is set and when it underflows, the sum is made zero. At the end of ϕ_4 the mantissa of the sum is in $E_{29}\sim E_9$, the exponent in $E_9\sim E_2$ and the sign in E_1 (Fig. 12). When the transfer time continues and the augend

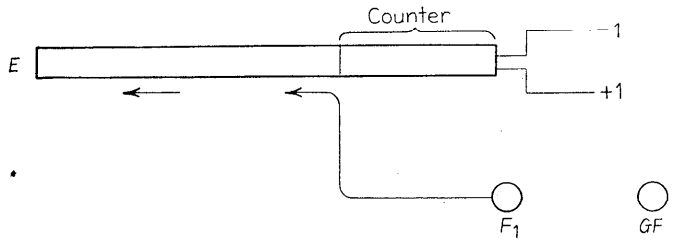


FIG. 12. ϕ_4

comes every four word time from the line L, the summation of floating point numbers is produced in the register E.

(VI) The floating point multiplication also takes four word times which are written ϕ_1, ϕ_2, ϕ_3 and ϕ_4. In ϕ_1 of (VIa), just as in (IVa), the absolute value of the mantissa of the multiplier gose into the register E, that of the multiplicand goes into the line A, delaying one word time. The sign of product is set in E_1 at t_1 time. IG_1, IG_2 detect zero value of both numbers and the bit showing zero product is stored in F_1. At $t_{ex}\cdot\phi_1$, the sum of the exponents $\xi+\eta$ is made by passing through the full adder FA_0 and stored in $E_9\sim E_2$ where it is corrected in the excess 2^7 form at $t_1\cdot\phi_2$. If the exponent exceeds 2^7-1, the overflow indicator is set, if it is below 0, F_1 is set for zero product.

From the bit time $t_{10}\cdot\phi_2$, the mantissa of the multiplicand, coming from the line A and its double and triple values are selected by the quaternery digits of the multiplier in $E_{29}\sim E_{10}$, adding them though the distributed full adders into the shifting register F, making the 40 bit length product. For normalizing, at $t_7\cdot\phi_3$, when 22 bits in the high order of the product are in $F_{23}\sim F_2$, if F_{23} is zero, the right shift is stopped one bit time and the exponent in $E_9\sim E_2$ is subtracted by one. For rounding off to make 20 bits mantissa, the carry flip-flop CF_1 of the left most full adder FA_1 between F_2 and F_1 is set at $t_8\cdot\phi_3$ and the right shift is stopped at $t_8\cdot\phi_3$, when the round off carry is stored in CF_1. These right shift operation is directly controlled by the flip-flop GF as before. In ϕ_4, if $F_1=0$ which shows the product is not zero, at t_1 the sign bit in E_1, at t_{ex} the exponent in $E_9\sim E_2$ and at t_{man} the mantissa from the sum terminal of FA_1 are sent to the destination line via the late bus (Fig. 13).

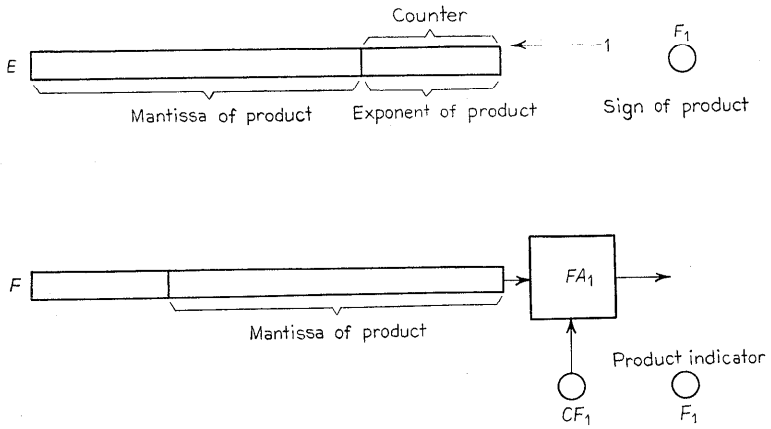


FIG. 13. Floating point multiplication

In the case (VIb) and (VIc), the flow is same as in (IVa) and (IVc), but the process is as stated above. $A^n\times B$ is constructed in $4n$ word times instead of $2n$ word times in the fixed point operation.

(VII) In maximum value selection operation, the numbers in the line L flow into E or F register so that the maximum absolute value in L remains in E register and its address in the line A. Data from L go to the full adder FA_0 and to the F register via EB_1 and IG_1, where their absolute value are taken. At first the number in the E register is assumed maximum and set $GF=0$. Then the data in E recirculate in itself and its inverse goes to the full adder FA_0 making comparison with the data from L. The carry bit of FA_0 at t_{29} determines which value is greater. If it is zero, the data in E is greater and GF remains zero

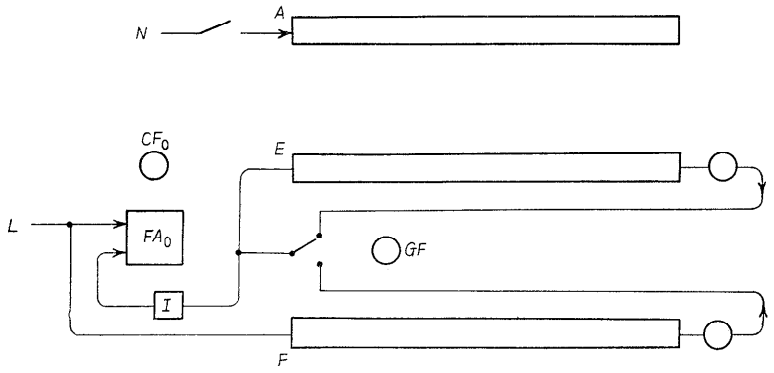


FIG. 14. Maximum selection

for the next one word time, so that the new comparison cycle is repeated same as above. If it is one, the data from L is greater, GF is set one and the data in F is used for the next comparison insted of the data in E, which is replaced by the data in F during comparison process, and the line N is connected to the line A for that word time. Thus at the end of this block operation, the maximam value in the line L is in E register and its address in the line A.

(VIIb) Data from L go into E register via EB_2, IG_2 where non zero is tested and the recirculation is stopped. Non zero word detection make $GF=1$ for the next one word time when the recirculation of the line L is restarted and the address is recorded from the line N into the line A. With the interative use of this command, only non zero data can be selected without testing for zero every words in a array and program branching.

(VIIIa) This is just the "and" operation on the way of the data from L and L' via EB_1, EB_2, and LB to the destination line.

(VIIIb) One bit delay is inserted in the circulation except for sign bit time. Zero is written at the bit time t_2.

(VIIIc) This is to make the line L one bit short and recirculate except sign bits which go back normally. At t_{29}, zero is written in the line L.

(VIIId) Double length numbers recirculate in the line L. The recirculation is inhibited at the even word time and sign bits, delayed one word time via the line A, are written in the line L at t_1 of the odd word times. Thus a group of double length words are changed to a single length word group.

Those stated above are the explanation of the flow control of block operations. There are many other useful block operations, but they can be synthesized by combining the block operation commands referred above.

5. *Examples of Programs*

Build-in block operation commands make processing faster as well as programming easier. A few example are shown.

(I) a_i's, b_i's are given in the consecutive addresses in the line L and L', $\sum a_i b_i$ is calculated as follows.

1. clear line B. (two word line for $\sum ab$)
2. make $\sum a_i b_i$ in line B. (i=odd)
3. shift one work length in line L ($i \rightarrow i+1$)
4. shift one word length in line L' ($i \rightarrow i+1$)
5. make $\sum a_i b_i$ in line B.

Thus $\sum_i a_i b_i$ is produced in the line B within five circulation times of the line L. When data are placed every other locations, an inner product can be made within one circulation.

(II) a_i's are placed in consecutive odd addresses. $\sum a_i x^i$ is calculated as follows.

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

Thus the polynomial $\sum a_i x^i$ is evaluated within two circulation times.

(III) Rearranging positive numbers a_i's ($i=1, \cdots, 100$) in decending order, whose initial order is random in line L.

1. select maximum value of line L into E register and place its addrss in line A.
2. test E register for zero, if zero, stop, otherwise go to next.
3. clear the location in line L, indicated by the content of line A. (This is not a single command but the clear command is made in line A by adding the constant and then line A is made command source. This command making process and its excution can be performed within one circulation time.)
4. circulate line L via register E for one circulation time. (every words of the line L is shifted left one word and content of E is stored in the most right word position of L)
5. jump to the command 1.

This sorting program can be completed within 300 circulation time. These are some of examples best suited for block operations, so that their speed is hundreds times higher than that with no block operation device. The report[2] says that in the problem of large matrices and floating point vectors, the solution time reduced to one sixtieth of that in which block operations are not used.

6. *Conclusion*

When dely lines are used for memories and registers, block operation technique, many of which are described in this paper, can easily be applied, to obtain high processing speed and easy programming. This idea were widely used in the two computers of Japan National Railways, which have been successfully operated more than one years. Though serial and extremely time sharing operations with 100 KC clock rate, they compete with large scale high speed computers for special types of works. These technique can be applied to other special purpose as well as general purpose computers. If higher clock rate as with magnetic strictive delay lines, are used, the operation speed is proportionally raised.

Auther wishes to express his gratitude to the members of the Automatic Control Laboratory at the Railway Technical Research Institute of Japan National Railways, who descussed and helped this project, and to those who worked for the construction of the computers, at the Electronics of Division of Hitachi Ltd. and at the Research Laboratory and the Electronics Division of Mitsubishi Electric Co.

REFERENCE

(1) M. HOSAKA, Y. OHNO, T. TANI: Seat Reservation System of Japan National Railways, Proc. Symposium on Data Processing Machines, Denki Tsushin Gakkai, Oct. 1959.

(2) S. SUZUKI: Matrix Calculation, The first symposium of Sūri Kakaku Sōgō Kenkyu, group 4. Jan. 1960.