THE UNIVERSITY OF MICHIGAN

COLLEGE OF ENGINEERING
Department of Electrical Engineering
Information Systems Laboratory

Technical Note

A MULTI-LAYER ITERATIVE CIRCUIT COMPUTER

Rodolfo Gonzalez

ORA Project 04794

under contract with:

UNITED STATES AIR FORCE
AERONAUTICAL SYSTEMS DIVISION
CONTRACT NO. AF 33(657)-7391
WRIGHT-PATTERSON AIR FORCE BASE, OHIO

administered through:

OFFICE OF RESEARCH ADMINISTRATION ANN ARBOR

August 1962

TABLE OF CONTENTS

SUMMARY

A multi-layer iterative circuit computer is described which is capable of dealing with problems involving spatial relationships between the variables, in addition to the inherent multiprogramming capabilities of this type of machine organization.

Some of the novel features presented are:

(1) A path-building method which retains the short time access characteristic of the common bus system, while still permitting the simultaneous operation of several paths in the network without mutual interference. Furthermore, the connecting method allows communication between the modules in a one-to-one, one-to-many or many-to-many way.

(2) A specialization in the functions of the individual layers, separating the flow of control signals from the flow of information. This step-by-step treatment of the instructions makes it possible to pre-interpret them before the actual execution, thus permitting the inclusion of instructions acting from many-to-many operands.

(3) Three-phase operation, with each phase active simultaneously in each layer, and operating on different instructions. Due to the overlapping of the phases in time, the total effective time per instructions remains the same. Once an instruction has been executed, the partially processed results are transferred to the next layer, and the now vacant layer starts the processing of the next instruction.

The notable properties of the iterative structure are thus agumented by the inclusion of these features, resulting in a machine with iterative computational structure that includes a form of control or interpretation unit.

1. INTRODUCTION

A study of the organization of the latest large scale computers shows a trend to an ever increasing complexity from the system design point of view. This evolution towards complex systems has been dictated by the desire to increase the power of the computers, sometimes in the productivity aspect, and in a few other cases in the computational capability aspect.

Most machine designs have had as a goal the maximization of the use factor of the computer, or at least of the most expensive units, generally the fast memory. This has been achieved by resorting to input-output buffering, by incorporating multiprogramming facilities reducible in the last analysis to time sharing procedures, by the inclusion of partial multiprocessing capabilities, and in some cases by creating a programmable structure organization as in the polymorphic machine.

It must, however, be recognized that most of the available commercial machines tend to maximize the productivity, that is, they try to minimize the cost per instruction, which is proportional to the ratio of speed to cost per operation.

On the other hand, very little has been achieved with respect to increasing the bounds of practical computability. Thus the problems encountered in the fields of pattern recognition, games, simulation and adaptation still need a computer capable of handling them in an efficient manner.

The iterative circuit computer has been considered as the most suitable solution for these types of problems which have in common the characteristic

that the spatial distribution of the modules is an homomorphic image of the relations governing the interaction of the variables. The undisputed suitability of this class of computers for these problems has relegated to a second place some other properties of the iterative structure that are not exclusive of this organization, but which are much more easily implemented in it than in a system with specialized units. The iterative circuit computer provides the possibility of true simultaneous multiprogramming, plus the powerful resource of infinite interaction between the programs. Neither of these characteristics is present in any of the more sophisticated systems now available.

Furthermore, an individual module can function at various times as an accumulator, register, memory cell or simply as a connecting link, and can be activated in any of these functions at any time during the execution of the program. Therefore, we are in the presence of an organization even more flexible than that of a polymorphic machine.[12] Although it would seem inappropriate to apply this term when there exists a lack of specialized units, it must be remembered that the modules are structurally alike, but their instantaneous functional behavior is different and is defined by the current instruction.

The polymorphic system is a programmable structure machine, but the changes in structure are performed on the interconnections between modules and not on the internal organization of the computer modules. Therefore, the full advantage of a changing structure is not realized, although a great increase in the use factor of the system is obtained. There are two

factors affecting the effectiveness of the system.

The first is due to the specialization and non-convertibility of the units, that is, there is a fixed number of components of each type available resulting in a limited number of combinations that can cover only a limited class of problems. Many problems cannot be handled efficiently because of the lack of more units of a certain type, while at the same time there may be a certain number of idle units that cannot be put into service because they perform different functions.

The second factor is closely related to the first, and is connected with the problem of priority assignment. It has been shown[13] that an attempt to obtain a high utilization factor for the computing modules increases the mean queue length. If there exists a number of programs with low priorities, then a high use factor can be obtained, but usually a compromise must be reached between efficient utilization of equipment and length of waiting lines.

In an I.C.C., however, any number of modules can be performing any one of the possible functions in one step of the program, and entirely different ones in the following operations. Also, the polymorphism of the machine is a function of the current instruction, not of the maximum requirements of the program.

The available literature on I.C.C.'s is surprisingly scant. References 1,2,3 cover the design considerations, both for uni-dimensional and two-dimensional networks. Reference 3 includes also the treatment of the problems of stability and equivalence of iterative networks. References 4,5,6 cover the specific problem of embedding a computer in the logical iterative net-

work. These are practically the only proposals for a computer based on this type of networks. Unfortunately, reference 4 covers only a special-purpose computer intended for pattern recognition and allied spatial problems. The paper by Holland[5] has been the starting point for a number of projects, but its title has mislead many into believing this was a proposal for a practical machine. While most of the ideas are worthwhile, they are by no means unique or optimum, as S. Amarel has clearly pointed out in his review.[8] Holland only describes a mathematical model of a space in which a simulation of the physical laws governing the interaction of a system with the environment can be set up. As such, the model possesses all the uniform properties and generality necessary for its use as a simulator in which the process of adaptation can be studied. While it still retains the power of an ordinary computer, its use as such would imply a wasteful employment of its potential capabilities while its performance would be hindered by an excess of non-essential features for this particular role.

Especially criticizable are the following features which affect characteristics that are fundamental in any I.C.C.:

The scheme used for selecting operands suffers as a consequence of both the method used for addressing and the path-building procedure necessary to reach them. The addressing method employs a "floating" reference, that is, all the addresses are relative to the address of the module active at that moment, and therefore an operand address assumes a different representation in every instruction that refers to it.

The path-building procedure has the disadvantage of being essentially sequential, resulting in a long effective access time, and therefore assigning great importance to the problem of data allocation.

These difficulties can very well be attributed to the lack of organs of command and to the circumstance that both control and information channels flow through the same network.

In Newell's paper,[7] however, we find the first reference to a multi-layer iterative structure, and furthermore, he suggests solutions to the problems of grouping modules to function as single entities and for the simultaneous selection of operands. It seems therefore logical to try to specify the organization of a multi-layer machine having each of the layers fulfilling some specialized function, yet being in itself a complete iterative structure.

The purpose of this paper is to present one possible example of such an organization, in which the following new characteristics are incorporated:

(a) A path-building procedure having the short-time access advantage of the common-bus system, but which also allows simultaneous multiple path building with no mutual interference.

(b) Three-phase operation, with specialized networks operating simultaneously in different phases on three consecutive instructions.

(c) A specialization in the functions performed by the stacked networks.

(d) Inclusion of geometrical operations in addition to the arithmetic and logical ones.

2. DESCRIPTION OF THE COMPUTER

The computer is composed of three stacked layers each consisting of an iterative network of $\underline{m}$ x $\underline{n}$ modules. The three layers are exactly alike in size, shape and type of modules used.

One layer is called the "program plane." This contains at the start the original program or programs, and later the modified programs resulting from the interaction of the original ones. Fig. 1.

The intermediate layer is called the "control plane" and its function is to interpret the instruction following the one being executed at the moment, determining the operand(s) and storing in them the full instruction. These "image operand(s)" in the control plane will in turn generate activation signals which will be transmitted on the wires connecting correlative modules and will determine which modules in the third plane will act as operand(s) II in the next phase.

The third layer is the "computing plane" where the actual arithmetic, logical and geometrical operations are performed.

The number and distribution of the modules active at any time is determined by the signals transmitted from the "control plane" in response to an "operation complete" pulse from the computing plane.

Communication between the three layers is provided by the following sets of connections. Fig. 1:

(a) A set of connections from every module in the program plane to every correlative module in the control plane.

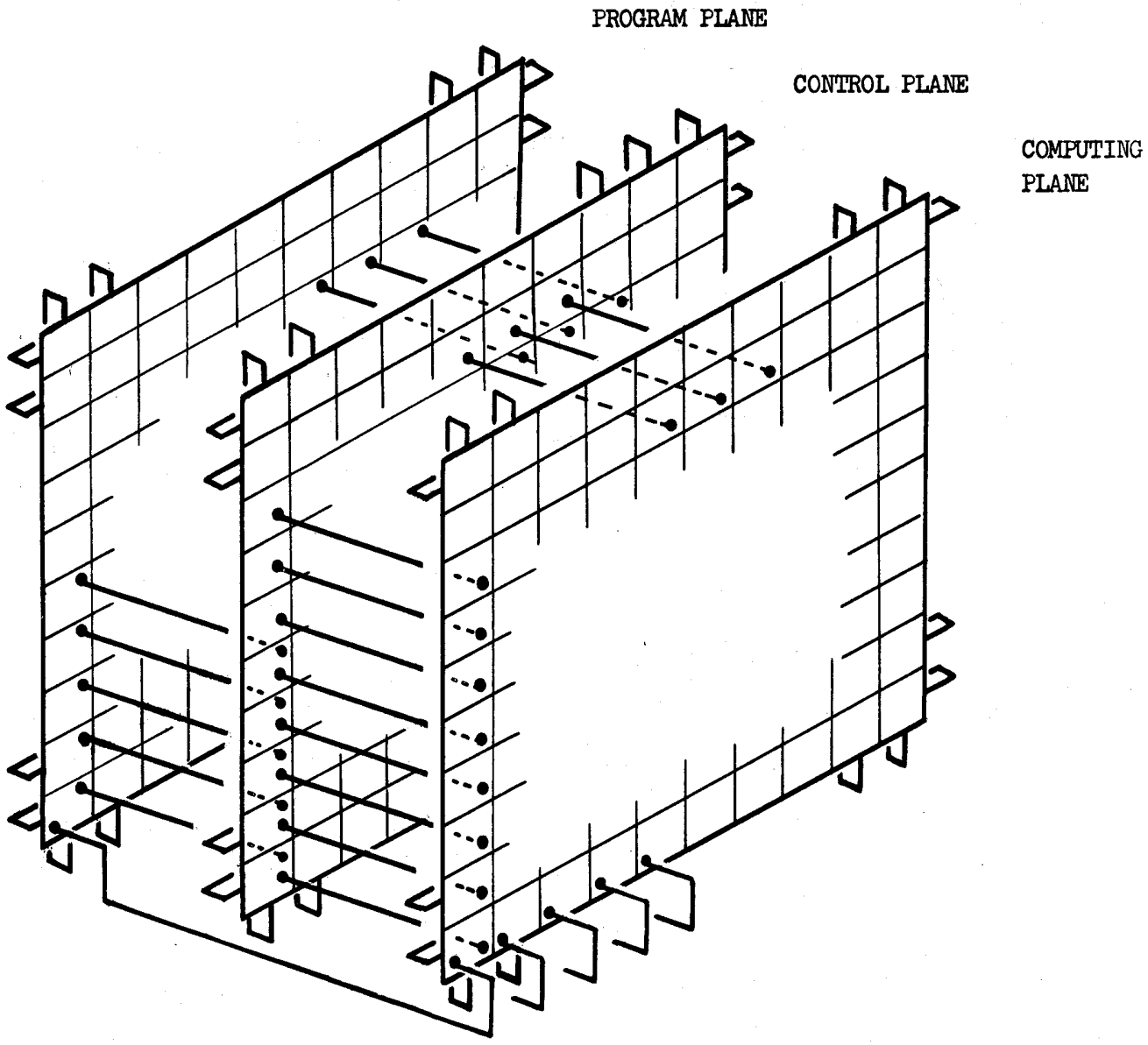

Fig. 1. Inter-layer and wrap-around connections.

(b) A similar network of connections from the modules in the control plane to those in the computing plane.

(c) A similar set of connections from the modules in the program plane to those in the computing plane.

(d) A common bus line connecting all the modules in the computing plane, and transmitting the "operation complete" pulse to all the modules in both the program and control planes. Fig. 2.

The connection lines described in (a), (b), and (c) are called activation lines.

3. DESCRIPTION OF THE PLANES

Each plane consists of a network of $\underline{m}$ x $\underline{n}$ modules, the modules being connected by a line called the information line running in each row and column through the normally conducting gates in each module. Fig. 4.

Besides these inter-modular connections, there exists an end-around connection between the first and last module of each row and similarly for the terminal modules of the columns. Therefore, there is a separate information line for each column and row, which is closed on itself by the end-around connection. These end-around connections provide the spatial continuity of the structure, transforming the planar distribution into one where a uniform neighborhood relation holds for all the modules, with no constraints due to physical boundaries. The resulting continuity provides the same behavior as that of a network spread over the surface of a torus.

4. DESCRIPTION OF THE MODULES

All the modules in the three planes are exactly alike. They communicate with each other in the same plane by means of the column and information lines running through them, and with the correlative modules in the other layers by means of connections called activation lines.

The internal structure of the modules includes an accumulator, a register, a decoder and several switching matrices to connect the former to the information lines. These units can be described as follows:

(a) An accumulator capable of performing addition, whose input is supplied from the output of a switching matrix connected to the four possible inputs to the module. The accumulator is connected through parallel gates with a register of the same length, which is described in (b).

(b) A register of the same length as the accumulator, and which is connected with it through parallel gates. This register simply copies the contents of the accumulator every time a load-type operation is completed. At the same time, it supplies the only output lead over which the contents of the accumulator can be read out. This means that every time some instruction requires the transmission of the contents of the accumulator, the information is actually taken from the corresponding register. The output can be directed to any of the module's four terminals by the switching matrix (d).

(c) A switching matrix with inputs from the module's four input terminals, and whose output is connected to the input of the accumulator. The

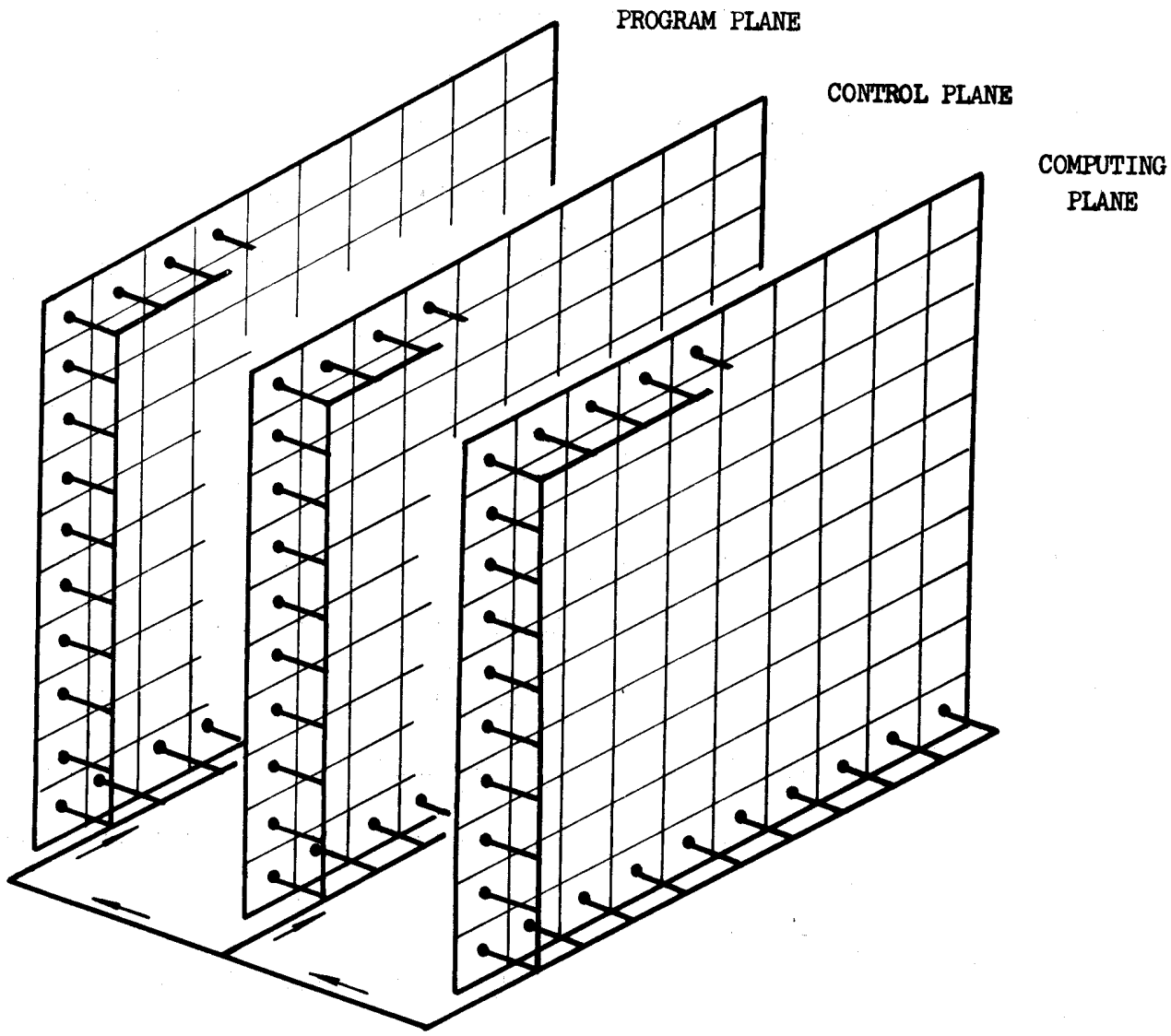

Fig. 2. Three-plane structure and common buses.

setting signals for the matrix are supplied by a decoder, described in (g).

(d) A similar switching matrix, whose input is the output of the register, and whose output feeds any of the four terminals.

(e) A pair of gates, normally conducting, that connect the terminals belonging to opposite sides of the module, thereby maintaining the continuity of the vertical and horizontal information lines across the module, with no connection between them.

(f) A switching matrix that can connect any input terminal to its immediate neighbor. In this way, a corner in the transmission path can be formed.

(g) A decoder, which receives the complete instruction on the normally continuous information line, and which compares the address in the instruction with its own address, producing output signals that govern the setting of (c), (d), (e), and (f).

(h) A similar decoder for the other information line.

(i) A gate connecting the output of the register with the activation line going to the input of the correlative module in the computing plane.

5. WORD FORMAT

A word is composed of four fields: the operand I field, the code field, the operand II field, and the successor field.

The operand I field contains two pairs of symbols; the first pair indicates the rows to which the first and last modules in the pattern or group of modules

belong; that is, it gives an indication of the extension of the pattern of operands I. The second pair does the same with respect to the initial and final column coordinates.

Example:

Operand I field	Code field	Operand II field	Successor field
(36;22)	Load	(22;33)	(33;77)

Since we are dealing with linear patterns, that is modules that are all in one column or row, at least one of the pairs in the operand I or operand II fields must contain a repeated number to indicate that only one column or row is involved. Example: (36;22) indicates the pattern extending from row 3 through row 6 and belonging to column 2. Therefore, the operand I field indicates which modules will be operands I when the instruction is executed. Similarly, the operand II field gives the address of the group of modules which will become operands II.

The successor field specifies the address of the location of the next instruction, and therefore is always of the form (XX;YY) since it must necessarily refer to a single module.

6. PATH-BUILDING PROCEDURE

The method used for communicating between modules in an iterative circuit computer is one of the key factors that determine the efficiency of the machine. The method described here is not strictly a path-building procedure since the

connections are permanently established as row and column information lines. The procedure only determines the operator and operand locations, and sectionalizes the corresponding row and column information lines into segments that are connected together at the cross-over point.

The general structure of the switching arrangement within each module is shown in Fig. 3. The gates are shown as bi-directional to simplify the diagrams. The row and column information lines run through all the modules in the corresponding row or column forming a closed loop since all the switches in the path are normally closed. Fig. 4.

The sequence of operations is as follows: The current instruction is stored in the active module, in this case the module at the upper left corner of Fig. 5. The instruction word is transmitted over one of the two information lines, the choice depending on the shape of the pattern of operand I. If the operands I are all in one column, then the information is transmitted on the row information line and vice-versa. Since we deal with linear patterns, one of the pairs of coordinates in the operand I field will always be of the form XX; the repeated number indicating that the pattern extends linearly over the X column or row. The instruction transmitted on the information line that spans the whole row or column where the operand I is located is received by the decoders in all the modules in that row or column. Each decoder checks for coincidences between the operand I and II addresses contained in the instruction and the corresponding addresses of its own module. This includes the module originating the information.

In Fig. 5, the operand I has the address (33;22) and the operand (55;66). When the decoders in the modules of the first column compare these addresses

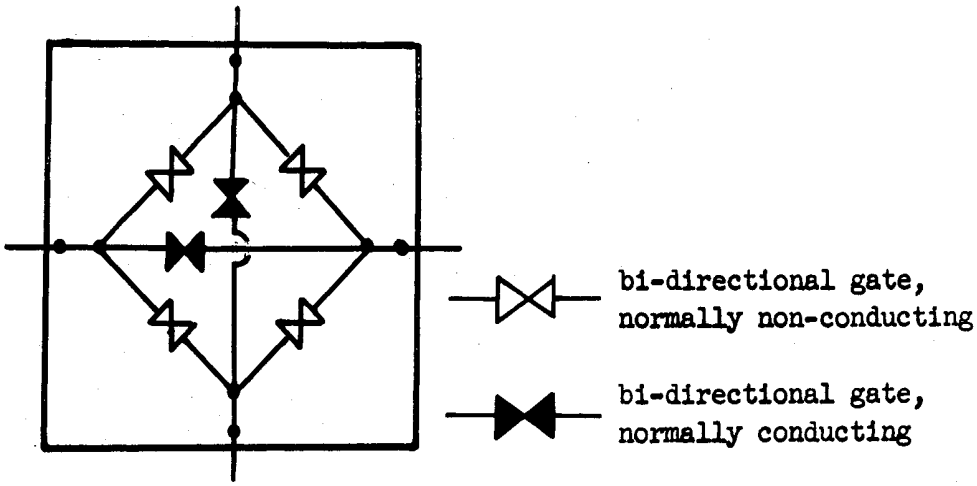

Fig. 3. Information-line switching in a module.

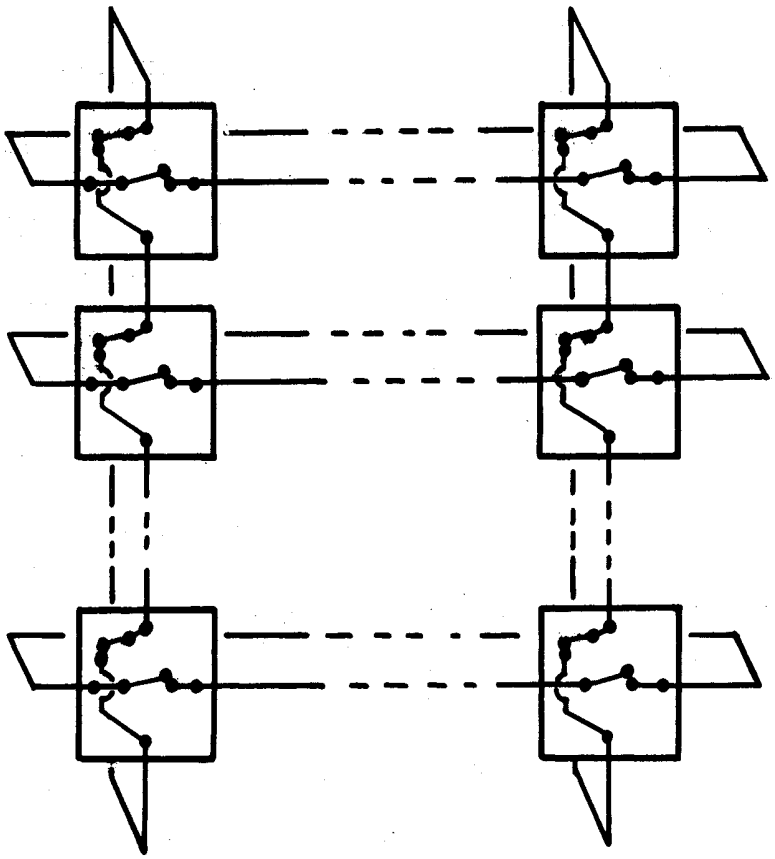

Fig. 4. Column and row information lines.

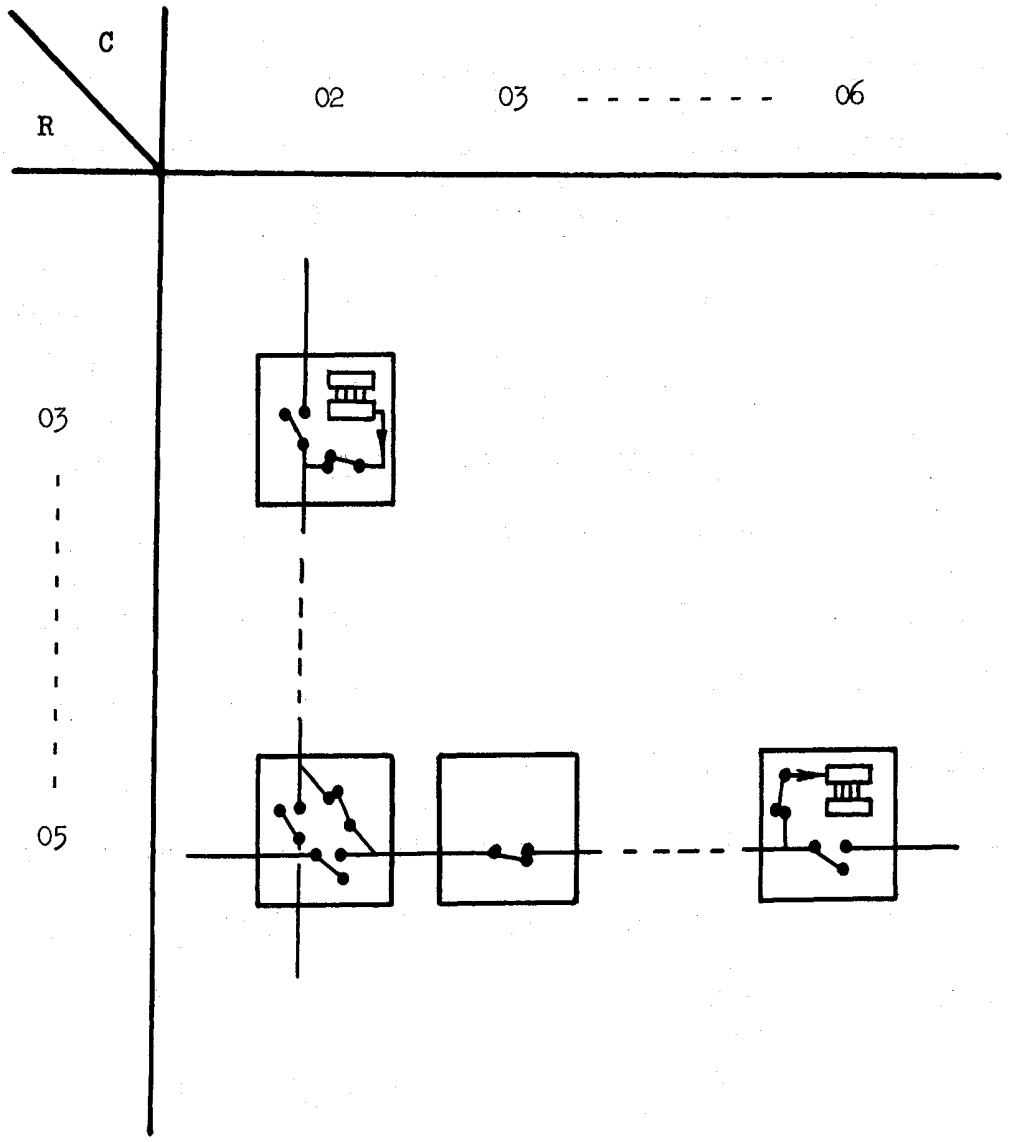

Fig. 5. Path connection for the instruction: (33;22)(store)(55;66).

with their own, two types of coincidence may arise:

(a) Double coincidence between the row and column coordinates belonging to one of the operand fields in the instruction, and the corresponding ones in the module address.

(b) Double coincidence between one row address in one field, one column in the other field and the corresponding ones in the module address.

It is evident that case (a) occurs only when checking the addresses of either the operand I or the operand II. In the first instance, the addresses in the operand I field will coincide with the addresses in the corresponding field in the module address. When the operand II is checked, the operand II field addresses will coincide.

The second case will occur at the module situated in the intersection of the column and row to which the operand I and operand II belong. In Fig. 5, the following situation will arise:

Instruction: (33;22) Store (55;66)

Intersection address: (55;22)

The double coincidence is between row and column addresses belonging to different fields in the instruction word.

The different results of the coincidence checking procedure are used to trigger two different sequences of events:

(A) If case (a) occurs, then either an operand I or II location has been reached, and the decoder activates one unit (e) and either (c) or (d), as described in Chapter 4. As a consequence, the following operations take place:

(i) The switch in the information line is opened, isolating the rest of the line.

(ii) Either the input or the output of the accumulator is connected to the information line. The operand I is always the source of information and therefore the transmission is from the operand I location to that of operand II.

(B) If case (b) occurs, then a corner in the path has been reached, and the decoder activates both (e) units and the (f) unit. As a consequence, the following operations take place: (i) Both switches in the row and column information lines are opened, completing the isolation of a piece of line from the terminal module to the corner in each information line. (ii) One of the switches connecting adjacent sides is closed, connecting the two isolated pieces of line and forming a continuous path from operand I to operand II.

The above procedure takes only two pulse times because all the decoding takes place simultaneously in all the modules in a row or column. Furthermore, it doesn't depend on the relative position of the modules to be connected, but only on the addresses of the operands.

In the case of instructions with multiple operands I and/or multiple operands II, it is possible to connect them in a one-to-one, one-to-many, or many-to-many way.

7. LIST OF INSTRUCTIONS

It is very common to speak of the operand I and the operand II when referring to an instruction, and usually no further distinctions are needed because there is only one accumulator. However in the case of the iterative circuit computer, where both operands are stored in modules that have the same capabilities, the distinction is no longer adequate. It then becomes

necessary to specify the direction in which information must flow since both modules can process the instruction and store the result.

Actually, any of the two modules could perform the role of accumulator and then we would have a left and a right instruction of each type, depending on which module executes the instruction and stores the result. In order to simplify the list of instructions, it is arbitrarily agreed that the operand II will always be the accumulator. Following this convention, the list of instructions for elementary arithmetic and logical operations is reduced to the following:

LOAD: Loads the contents of the location specified by the operand I into the location specified by operand II. The result appears in the location of operand II.

ADD: Adds the contents of operand I to the contents of operand II. The result remains in operand II.

COMPLEMENT: The contents of operand I is complemented.

TRANSFER ON NON-ZERO: If the contents of operand I is different from zero, control is transferred to the instruction located in the module specified by operand II. If the contents is equal to zero, the normal sequence of instructions is followed, that is, the next instruction executed will be the one specified by the successor field.

8. OPERATION OF THE COMPUTER

The execution of an instruction takes place in three phases, and each phase is performed in a different layer. Once a plane has executed its phase on an instruction, it waits until the next operation complete pulse from the computing plane causes the transfer of a new instruction to be operated on. Thus, the execution of the three phases proceeds simultaneously in the three layers, but with a different instruction in each layer. As a result, the effective operation time is one instruction per phase; the duration of the phases being determined by the computing phase being executed at the moment.

Fig. 6 shows the sequence of phases and the transfer of each instruction from plane to plane after the execution of each phase. The roman numerals indicate the phase, and the subscript the particular instruction being operated on. Thus, III_2 indicates that the second instruction is undergoing phase III. Phase III is the one that takes the longest time to perform and therefore is the one that generates the operation complete pulse that triggers the initiation of all phases in the three layers.

The sequence of operations that an instruction undergoes during the three phases is as follows:

PHASE I: The initiation of this phase is triggered either by an operation complete pulse or a start pulse. During this phase, the instruction following the one already in the control plane is made ready to be copied from the program plane onto the control plane in the same relative position. The net effect is to choose the successor to the current instruction already in phase

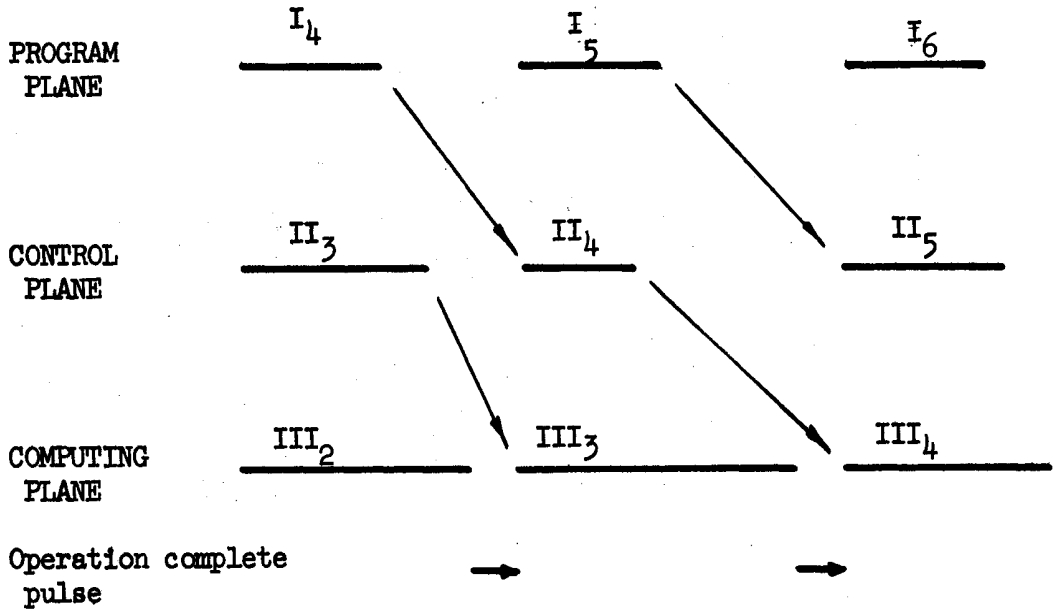

Fig. 6. Overlapping of phases.

II. The successor is specified by the address in the successor field and can be any location in the plane. In other words, it is not required that it be a contiguous neighbor.

PHASE II: The instruction activated in the program plane is copied in the same position in the control plane. In this plane, the operand II field of the instruction is interpreted to determine which modules are to be active (operands II) during the computing phase. Once the positions of the future operands II are determined, the whole instruction is transmitted and copied in these "image" positions in the control plane. The location of the "image" operators is found following a path-building procedure, as described in Section 6. Furthermore, the necessary data are copied from the correlative modules in the program plane. In this way, each future operand II position is now loaded with the instruction and the operand II itself.

PHASE III: The next operation complete pulse from the computing plane bus line initiates the third phase. The instruction and data now stored in the module or modules in the control plane are now transferred to the correlative modules in the computing plane, and each of these modules initiates a path-building procedure to connect itself to its operand or operands I.

At the end of this process, the modules containing the instruction are connected to their respective operands I. This connection can be from one module to another, from one to many, or from many to many, depending on the operation specified by the current instruction contained in the operands II. The locations thus selected receive the necessary data from the correlative positions in the program plane.

Once the connections have been established, the instruction is executed with information flowing in the correct direction. Operand I is always the source of information and therefore the output of its register has been connected to the information line. Similarly, operand II acts as the accumulator and the information line is connected to the input of its accumulator. The result is then transmitted to the correlative module in the program plane, where it is stored. Simultaneously with this activity in the computing plane, the control plane is now executing phase II on the next instruction, since it remains free once the "image" operators have been transferred to the computing plane. The completion of the execution phase is signalled by an "operation complete" pulse which is transmitted over the common bus from the computing plane to the similar buses in the program and control planes. This pulse initiates phase II in the control plane and phase I in the program plane.

This sequence of operations can be visualized following the transfer of instructions between the layers, in Figs. 7 through 10, while the computer executes the following sequence of instructions, supposedly part of a program:

1. (77;88) Load (44;55) (33;44)
2. (57;77) Add (57;33) (33;55)
3. (56;77) Load (66;23) (33;66)
4. (33;24) Load (66;22) (33;77)
5. (00;00) Clear(56;33) (22;77)
6. (55;66) T.Ø (33;99) (22;88) (Transfer on non-zero)
7. (00;00) No op(00;00) (22;99) (No operation)

Figure 7 shows the computer at the moment the first instruction is undergoing phase III. The state of the machine can be indicated by: III_1; II_2; I_3. The program plane is executing phase I on instruction 3, that is, it activates instruction 3 as the successor of instruction 2. In the control plane, both instruction 2 and the data corresponding to the operands II are being loaded into the locations asignated to the operands II. In the computing plane, instruction number 1 is being executed, with the contents of module (77;88) going into module (44;55).

Figure 8 shows the machine in the state III_4; II_3; I_2. The control plane is interpreting instruction 3, locating the positions of the image operands II, in this case the modules in row 6 and columns 2 and 3. Both instruction number 3 and the contents of modules (66;23) in the program plane are now copied into the correlative positions just determined in the control plane. The computing plane is executing instruction number 2, in this case, adding the contents of (57;77) into (57;33).

In a similar way, Figs. 9 and 10 show the execution phases of instruction numbers 3 and 4.

All instructions except the Transfer on Non-Zero instruction are treated in a similar way. The execution of instruction number 6, which is a Transfer on Non-Zero, gives an opportunity to explain in more detail the sequence of operations for this type of instruction.

When instruction number 4 is executed, an Operation Complete pulse is sent back to the program plane, and instruction number 6 is activated. It has to be remembered that instruction number 5 is already in the control plane.

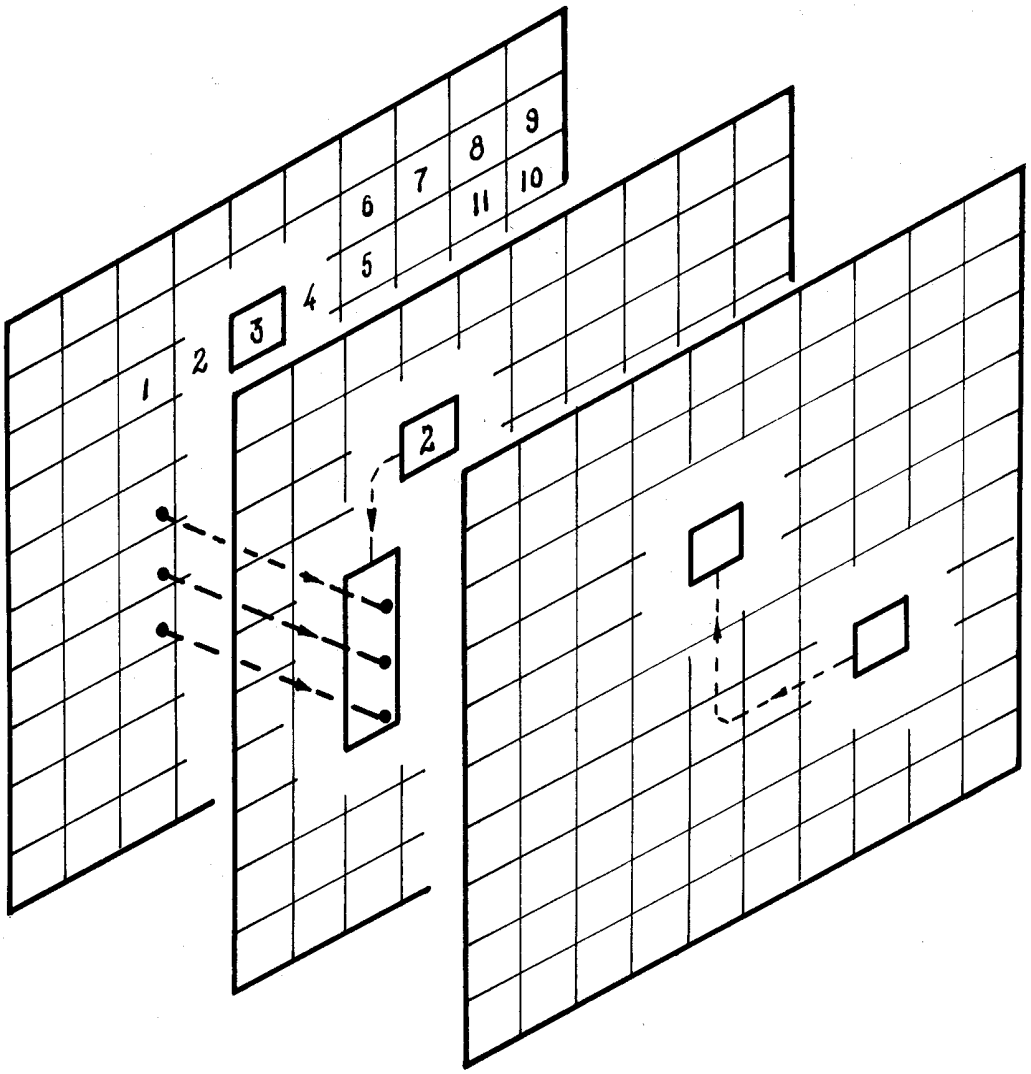

Fig. 7. Execution phase of instruction 1:
(77;88) Load (44;55) (33;44).

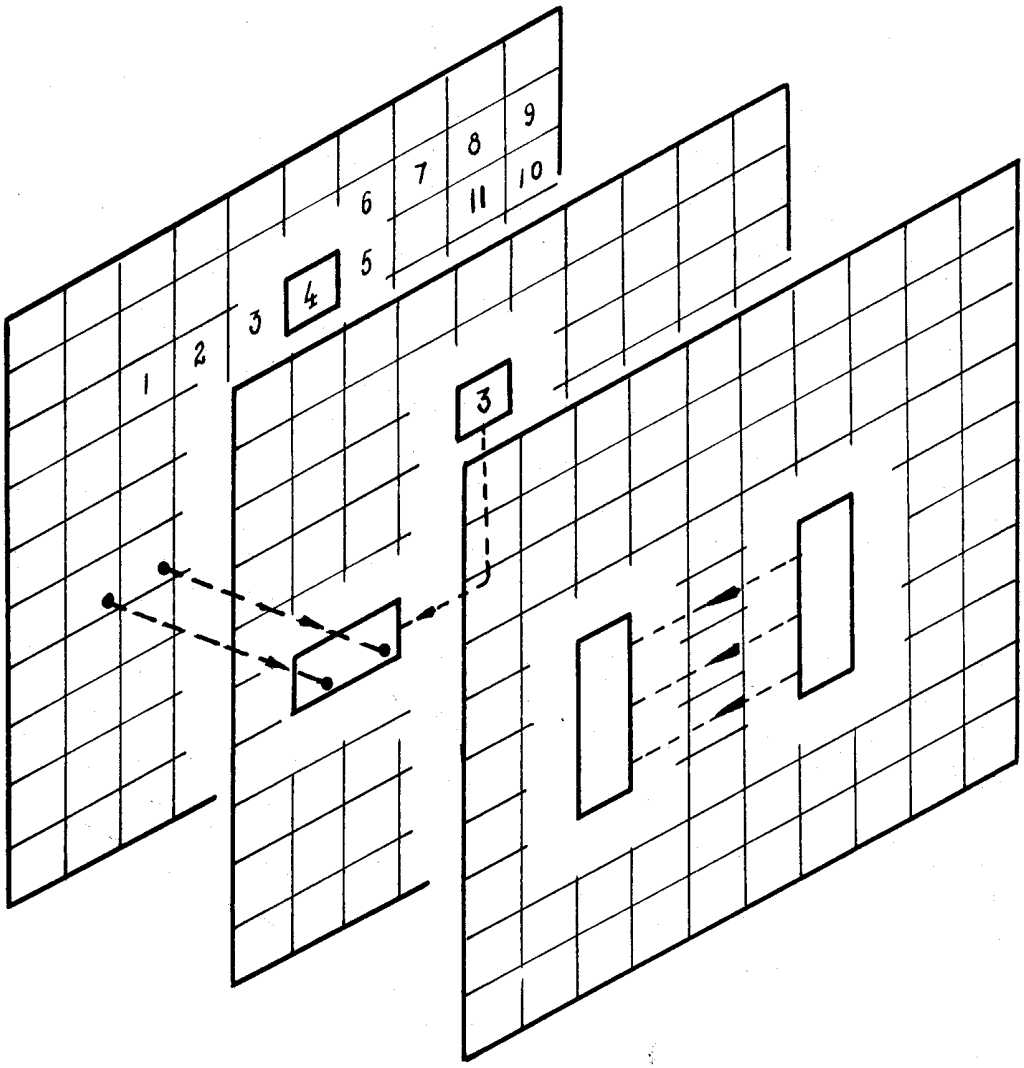

Fig. 8. Execution phase of instruction 2:
(57;77) Add (57;33) (33;55).

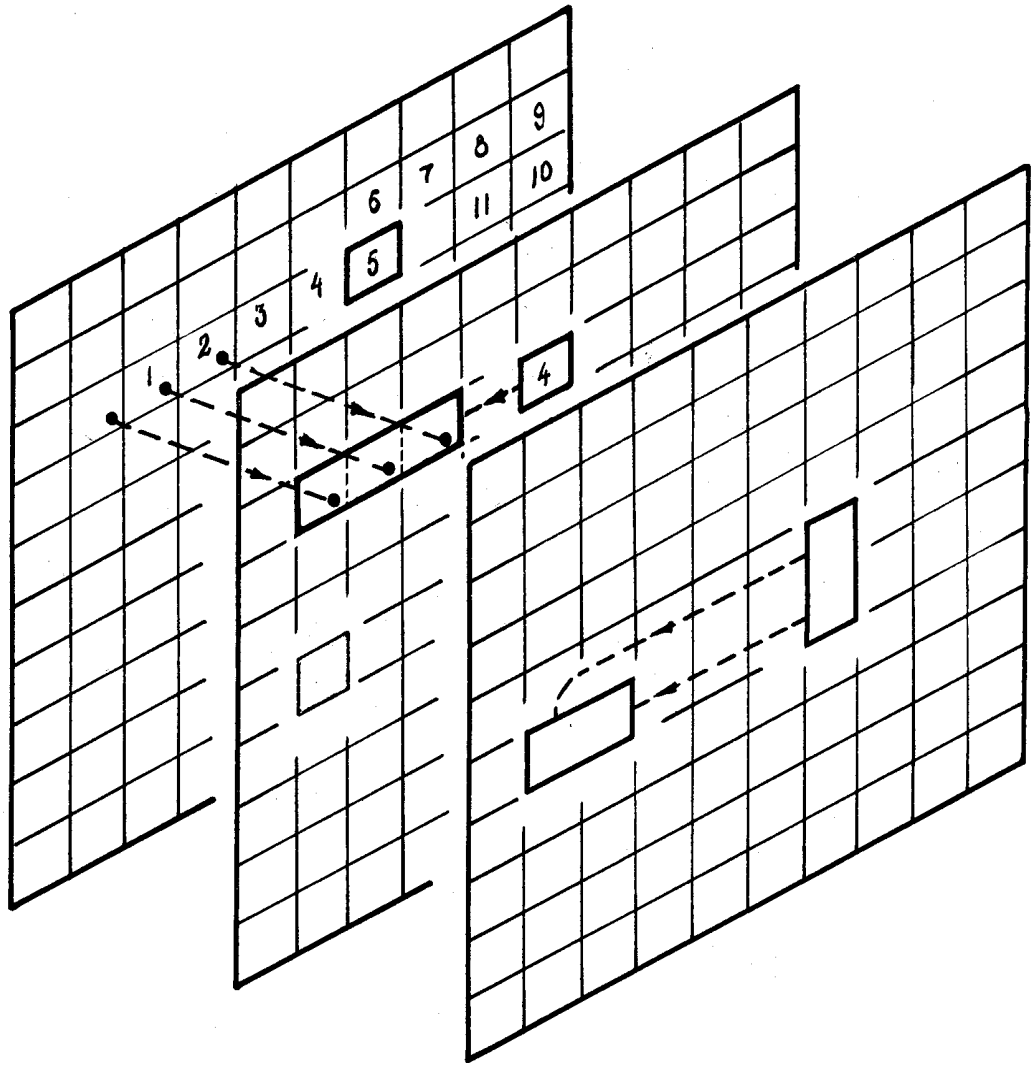

Fig. 9. Execution phase of instruction 3:
(56;77) Load (66;23) (33;66).

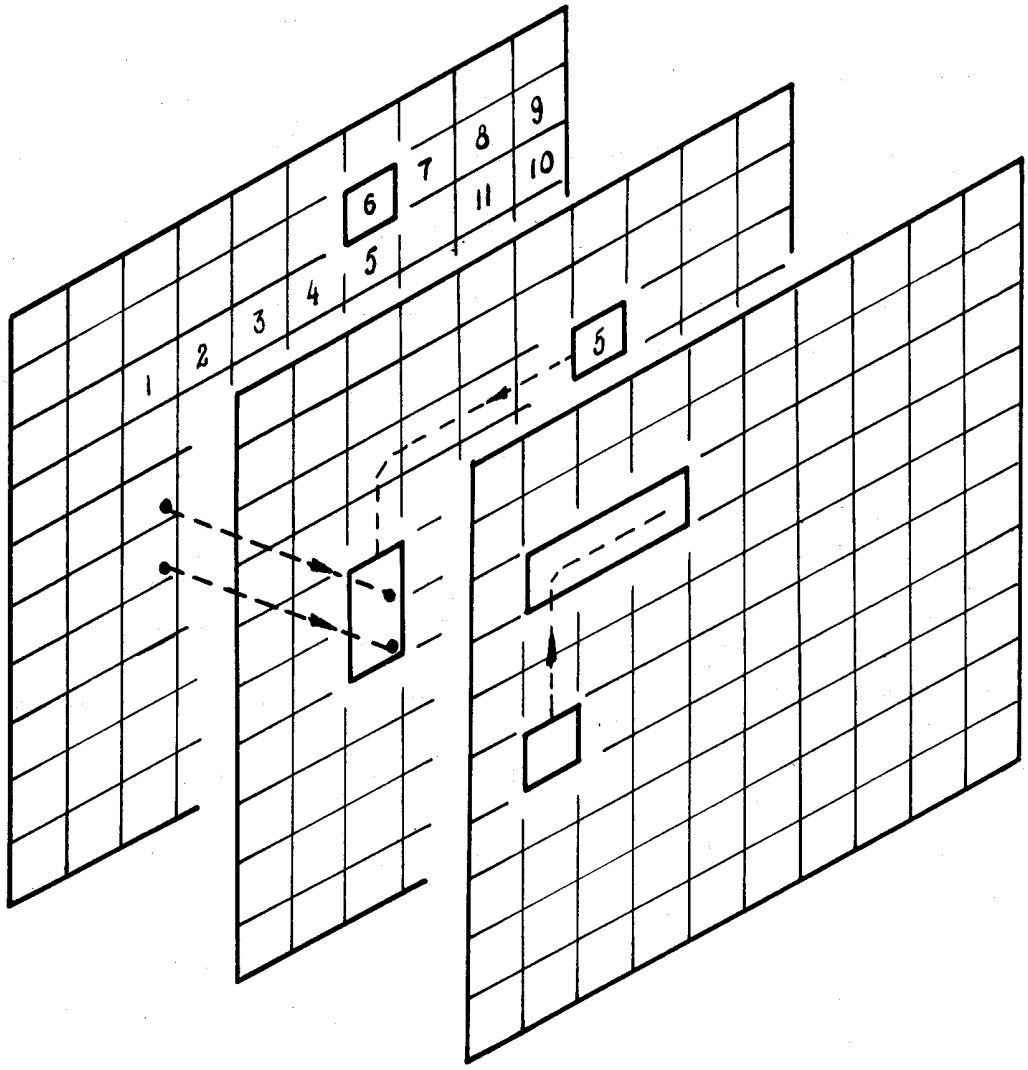

Fig. 10. Execution phase of instruction 4:
(33;24) Load (66;22) (33;77).

The situation is that illustrated in Fig. 10, and again in Fig. 11, but this time in a lateral view.

The operation complete pulse changes the situation to that illustrated in Fig. 12. Instruction number 6 is transferred to the control plane, and a path is built there connecting the module containing the instruction with the operand II, in this case module (33;99). The instruction is then stored in this module which receives the alternate address from the correlative module in the program plane.

The next operation complete pulse, signalling the termination of instruction number 5, produces a copy of module (33;99) in the computing plane. Fig. 13. This module is then connected to the operand I module, in this case (55;66). The operand I module contains the word of data on which the result of the transfer instruction depends.

The active module (33;99) then determines if the number in (55;66) is equal to zero or not. If the number is equal to zero, an operation complete pulse is emitted and the normal sequence of operations is resumed. If the number is not equal to zero, the active module (33;99) sends an activation signal to the correlative module in the program plane, activating it as the immediate successor and overriding the active status already obtained by instruction number 8. After a suitable delay, an operation complete pulse is emitted, and the normal sequence of operations is resumed. The delay is necessary in order to allow the newly designated successor to activate its own successor, which may be any position in the plane, not necessarily a contiguous neighbor. Fig. 14.

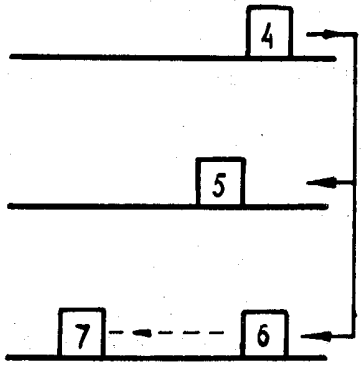

Fig. 11

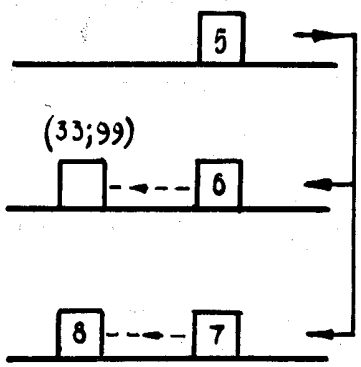

Fig. 12

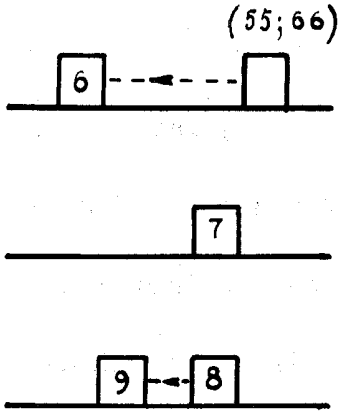

Fig. 13

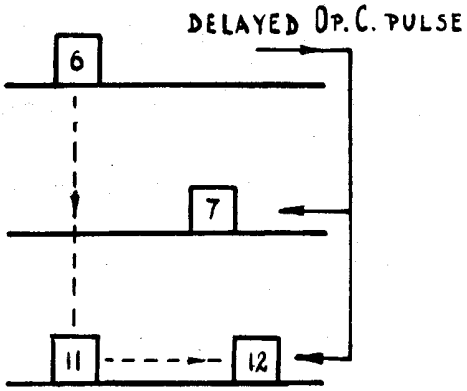

Fig. 14

Therefore, the sequence of instructions resulting from the transfer instruction is: 6, 7, 11, 12 ... instead of the normal sequence: 6, 7, 8, 9 ...

9. GEOMETRICAL OPERATIONS

When an attempt is made to process geometrical patterns in a computer in which the instructions refer to only two operands, it is necessary to divide the pattern into individual elements and operate on them one at a time. In the machine described here, the availability of multiple operand instructions reduces most of the geometrical operations to one of the arithmetic or logical ones.

In order to simplify the operation code, it is convenient to establish the relationship between the geometrical and arithmetic operations since most of the former can be interpreted as a particular case of multiple operand I and/or multiple operand II arithmetic operations. Thus, a Store operation can refer to a One-to-One (OTO), One-to-Many (OTM) or to a Many-to-Many (MTM) operation.

The OTO Store operation is the normal one, and in the geometrical interpretation would be called a COPY instruction.

The OTM Store instruction has two versions in the geometrical case: (i) The pattern of one module is to be repeated contiguously and linearly. Fig. 15. The corresponding geometrical operation is called EXTEND. (ii) The module has to be copied in repeated positions, all consecutive, but not contiguous to the original one. Fig. 16. The corresponding geometrical operation is called REPRODUCE.

The MTM Store operation repeats the pattern in a position parallel to the original one. Fig. 17.

The corresponding geometrical operation is called DISPLACE and reproduces the pattern in a parallel position. It implies a simultaneous one-to-one copy operation on many modules.

Therefore, a correspondence between the geometrical and arithmetic operations can be established, in which the first column can roughly be assimilated to a compiler language and the second one to a machine language.

COPY	Store OTO
EXTEND	Store OTM
REPRODUCE	Store OTM
DISPLACE	Store MTM

10. CONCLUSION

The organization presented here is not intended to be an ultimate design. Rather it presents one possible way of combining the intrinsic capabilities of the iterative structure with the advantages of an organization having some form of specialized control unit.

The principal advantage of the proposed organization resides in the fact that the multi-layer structure makes it possible to include a control plane which acts as a look-ahead unit, interpreting the instructions before the actual execution takes place.

This disposition provides the capability of dealing with instructions

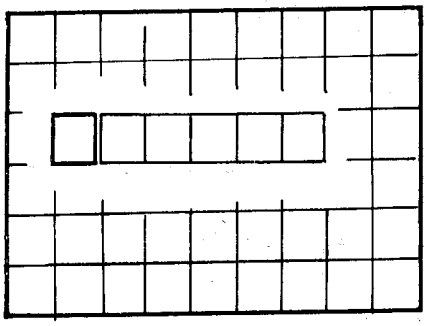

Fig. 15. EXTEND operation.

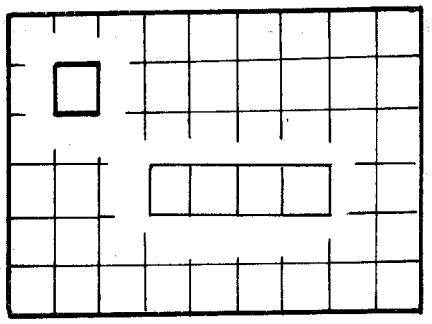

Fig. 16. REPRODUCE operation.

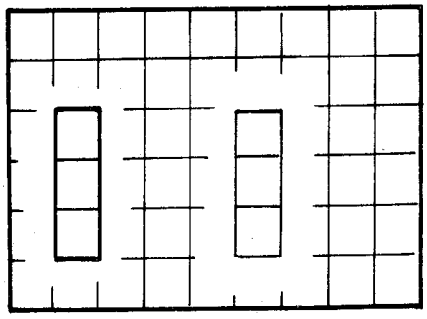

Fig. 17. DISPLACE operation.

that operate on any number of modules simultaneously, yet retaining in every step the possibility of true simultaneous operation of several programs with an unlimited degree of interaction between them. Moreover, the introduction of the look-ahead feature doesn't detract from the effective speed of computation, since the delay introduced by the pre-interpretation phase is compensated for by the overlapping of the sequence of phases which process consecutive instruction in different places simultaneously.

Furthermore, the method used for path building provides communication between the modules in the network with a very short access time.

Therefore, the combination of features given by the pre-interpretation of instructions and by the overlapping of phases can be regarded as a net advantage, with no penalty in time or complexity of the individual modules. The sole and inevitable penalty is the inclusion of two more layers. While this increases the number of modules by a factor of three, it must be remembered that the whole feasibility of this type of machine organization depends on the availability of components whose cost depends very weakly on the internal structure, and whose easy reproducibility assures a low cost per unit when used in large numbers.

REFERENCES

1. Henie, F. C., "Iterative Arrays of Logical Circuits," J. Wiley, N.Y., 1961.

2. McCluskey, E. J., "Iterative Combinatorial Switching Networks - General Design Considerations," IRE Trans. on Electronic Computers, Vol. EC-7, pp. 285-288, Dec., 1958.

3. Henie, F. C., "Analysis of Bilateral Iterative Networks," IRE Trans. on Circuit Theory, Vol. CT-6, p. 35, 1959.

4. Unger, S. H., "A Computer Oriented Towards Spatial Problems," Proc. of the IRE, Vol. 46, p. 1749, Oct., 1958.

5. Holland, J., "A Universal Computer Capable of Executing an Arbitrary Number of Sub-programs Simultaneously," Proc. EJCC, p. 108, Dec., 1959.

6. Holland, J., "Iterative Circuit Computers," Proc. WJCC, p. 259, May, 1960.

7. Newell, A., "On Programming a Highly Parallel Machine to be An Intelligent Technician," Proc. WJCC, p. 267, May, 1960.

8. Amarel, S., Review of (5) and (6), IRE Trans. on Electronic Computers, Vol. EC-9, p. 384, Sept., 1960.

9. Bauer, W., "Horizons in Computer System Design," Proc. WJCC, p. 41, May, 1960.

10. Squire, J. "A Comparative Study of Module Communications," Internal Report, Information Systems Laboratory, Univ. of Mich., 1962.

11. Carroll, A. B., and Confort, W. T., "The Logical Design of a Holland Machine," Internal Report, Dept. of Elect. Eng., Univ. of Mich., 1961.

12. West, G. P., and Koerner, R. J., "Communications Within a Polymorphic System," Proc. WJCC, p. 225, Dec., 1960.

13. Carlsen, R. A., Feingold, M. G., and Fife, D. W., "A Simulation of the AN/FSQ-27 Data Processing System" RADC-TR-61-254, Dept. of Elect. Eng., Univ. of Mich., 1961.