2-1974

Macromodular Computer Design, Part 1, Volume 1, Overview of Macromodules

Computer Systems Laboratory, Washington University

MACROMODULAR COMPUTER DESIGN PART 1. DEVELOPMENT OF MACROMODULES

VOLUME I

OVERVIEW OF MACROMODULES

Technical Report No. 44

FINAL REPORT - FEBRUARY, 1974
CONTRACT SD-302 (ARPA)

COMPUTER SYSTEMS LABORATORY
WASHINGTON UNIVERSITY
ST. LOUIS, MISSOURI

MACROMODULAR COMPUTER DESIGN

FINAL REPORT - CONTRACT SD-302

FEBRUARY, 1974

Technical Report No. 44

PART 1 - DEVELOPMENT OF MACROMODULES

VOL. I - OVERVIEW OF MACROMODULES

This work has been supported by the Advanced Research Projects Agency of the Department of Defense under Contract SD-302 and by the Division of Research Facilities and Resources of the National Institutes of Health under Grant RR-00396.

Computer Systems Laboratory
Washington University
St. Louis, Missouri

ABSTRACT

This volume contains a Foreword to the Final Report, two excerpted reprints that present the initial conception of macromodules and means for implementing them as seen in the early days of the project, and a third reprinted report that presents a summary and overview as of the autumn of 1972.

FOREWORD TO FINAL REPORT

This set of documents represents an attempt to bring together in one place sufficient material to enable the reader to obtain a reasonable overview of the major ideas and conceptions that gave rise to the macromodule development project at Washington University, and to report and record some of the details of the ensuing development effort and its results. Since the project is continuing with other support past the termination of the contract for which this is the final report, certain aspects of this report deal with efforts that are still in progress, particularly the design of restructured macromodules and the use of macromodules as a tool for technological and scientific research.

Part 1 of this report deals with the development of Phase I macromodules, of which over 800 modules of 17 types have been constructed and made part of a working inventory that resides at Washington University. This volume contains two excerpted reprints that present the initial conception of macromodules and means for implementing them as seen in the early days of the project, and a third reprinted report that presents a summary and overview as of the autumn of 1972. Further details of the functional definition and design of macromodules are presented in the remainder of Part 1.

The second Part of this report contains detailed manufacturing descriptions of macromodular parts and assemblies; it is based directly upon documents used to carry out the fabrication. This highly detailed material is intended to serve as a reference for those wishing to understand more thoroughly the material that is presented in Part 1, and may also contain useful ideas for other designers.

A status report on restructured macromodules makes up the single volume of Part 3 of this Report. Some of our judgments based upon our experience with Phase I macromodules are also reflected in that volume.

It should be pointed out that in addition to this report, there are numerous other documents dealing with particular aspects of the macromodular design project. These include a series of formal Technical Reports of the Computer Systems Laboratory, and an informal series of Technical Memoranda that were primarily intended for internal use and record purposes but contain substantial information of possible broader interest. Relevant documents of these series, as well as a bibliography of relevant publications in the open literature, are listed in Volume V of Part 1. Copies of relevant Technical Reports have been supplied to the Defense Documentation Center and should be available through their channels.

There is no adequate way to acknowledge and summarize the diverse and often intense interactions of people and ideas that have taken place during this project, and the precise roles and contributions of the participants in this large effort were often unclear at times even to the individuals who were directly involved. At this time it appears impractical to improve upon the acknowledgements that may be found in the reprinted articles contained in this volume, and a comprehensive but undifferentiated list of persons associated with the project since its inception is included in Volume V of Part 1. It is my sincere hope that their participation in this large effort will prove to have been as beneficial to them as it has to the progress of macromodular design.

Charles E. Molnar
Director
Computer Systems Laboratory
Washington University

A MACROMODULAR APPROACH TO COMPUTER DESIGN

A Preliminary Report

(reprinted here in its entirety)

Wesley A. Clark
Mishell J. Stucki
Severo M. Ornstein

Technical Report No. 1
February 21, 1966

Computer Research Laboratory
Washington University
700 South Euclid Avenue
St. Louis, Missouri

This work has been supported by the Advanced Research Projects Agency of the Department of Defense under contract SD-302 and by the Division of Research Facilities and Resources of the National Institutes of Health under grant FR-00218.

A MACROMODULAR APPROACH TO COMPUTER DESIGN

INTRODUCTION

The amount of logically irrelevant engineering detail inherent in the design and construction of a computer system is great. As a result, the task of creating a system based on the use of present techniques is so difficult and time-consuming that the number of different systems that can be put into use for evaluation and study by any one group of workers is small. This is unfortunate as we are thereby denied the opportunity to develop that insight into logical organization which can grow out of a working familiarity with many diverse forms. What is needed is a set of relatively simple, easily inter-connected modules from which working systems can readily be assembled for evaluation and study. With such a set, both the designer and user would be able to try out potentially powerful and novel structures on a very large scale, adjusting and improving the systems as needed. Once a design has been realized and its value established, it could then be reworked into tighter engineering form for maximum efficiency and for production by automatic wiring and fabrication techniques, and the experimental units made available for further studies or returned to "inventory" in the manner proposed by Estrin[1].

The modules as described are primarily vehicles for experimental use and as such must meet a set of requirements heretofore unnecessary in digital modules. Logical flexibility and ease of use must be considered of primary importance while factors such as operating speed, economy, etc., though not ignored, must be considered of secondary importance. The requirements can be summarized as follows:

1) The modules must be functionally large enough to reduce logical detail by a significant amount and must be relatively easy to understand and assemble. The number of different types should be small as possible so as to limit inventory, but at the same time, the set must be logically complete so that whole systems may be assembled. There must be not only central processor modules such as register and memory units, but also modules for power, signal conditioning,

input-output buffering and control, together with a reasonable selection of input-output devices themselves.

2) The mode of combining units into larger structures must be very simple (a problem first considered by Babbage, who examined this matter "with unceasing anxiety" one hundred and twenty years ago[2]). The modules should be designed for easy mechanical assembly. Communication from one mechanical assemblage to another should be accomplished by means of easily connected cables.

3) All units should be designed so that the assembling of these units into a working system presents no logically irrelevant problems such as those relating to circuit loading, waveform deterioration, signal propagation delay, power supply interactions, and so forth, regardless of the size or complexity of the system. The modules should be powered and perhaps cooled individually, and all possible signal paths must be provided with signal-standardizing amplifiers capable of driving all possible loads.

We call units which meet these requirements macromodules to distinguish them from the more conventional computer system modules. In this report we present a set of macromodules which, although not "complete" in the above sense, meets all other requirements and is sufficient for the synthesis of all central processor functions of which we are presently aware. Particular attention is given to the problem of control structures and a technique is presented in which the control signals for a given process are routed along a control network whose topology is isomorphic to the flow diagram representing the process. The step from conceptualization to realization can therefore be made directly, a situation that enormously simplifies design.

MACROMODULAR SYSTEMS

General Characteristics

The macromodules to be described are relatively small, dimensionally modular, structurally self-sufficient boxes which contain all of the required electronic circuits and memory elements. Electrical connectors on the faces of each unit provide all power and signal access. The units can be interconnected mechanically and electrically to form larger assemblages, and standardized cables are provided for all inter-assemblage communication. All connectors are backed by signal-standardizing amplifiers capable of driving any attachable module or cable.

Data processing modules are organized in parallel binary form with a word-length modulus of 12 bits, and are designed functionally for asynchronous operation. Memory modules hold 4096 12-bit words.*

The design of a system based on these modules requires, we believe, only the exercise of logic. The operability of the resulting system cannot be adversely affected by the physical distribution or arrangement of parts, the distances between units, the number or diversity of modules, or the routing of the interconnecting pathways. Macromodular systems are, as a result, capable of continuous growth and functional enrichment.

* The numbers 12, 4096, and other such parameters have been made specific, for purposes of this report, only to simplify description.

System Organization

Macromodular systems may be viewed in terms of two logically distinct, interacting networks (Fig. 1). The Processing Network (the heavy-lined structure) consists of data processing elements interconnected by data pathways, and provides for the storage, propagation, and transformation of data within a system. The Sequencing Network (the light-lined structure) consists of control nodes distributed throughout the system, interconnected by control pathways. The structure of the Processing Network defines the basic data processing operations of the system while the structure of the Sequencing Network defines the order in which subsets of these basic operations can be carried out.

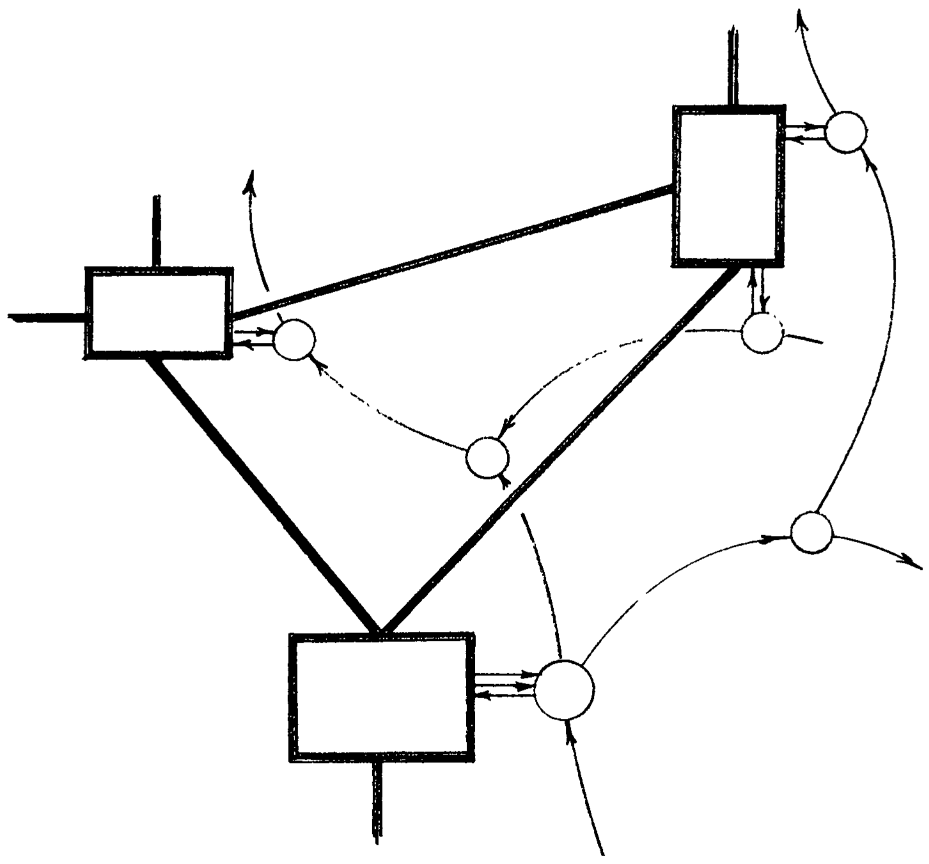


Fig. 1

Interaction between these networks takes place at control terminals on the data processing elements. These terminals have two functions: they allow the Sequencing Network to initiate operations, and they return completion signals when the operations are finished. Each basic data processing operation has an associated set of these terminals (Fig. 2), the number of terminals in the set being determined by the nature of the operation. Operations that manipulate data, <u>data operations</u>, have two, an initiation terminal and a completion terminal. Operations that check data for specific values, <u>decision operations</u>, have more than two, one to initiate the operation and the others (completion terminals) to indicate the value of the data found. Also shown in Fig. 2 is a <u>time-continuous transformation</u> element. This element, unlike those already described, performs its operation continuously. The data presented at its output changes only in response to changes of input data rather than in response to control signals, and as a result, the element has no control terminals at all.

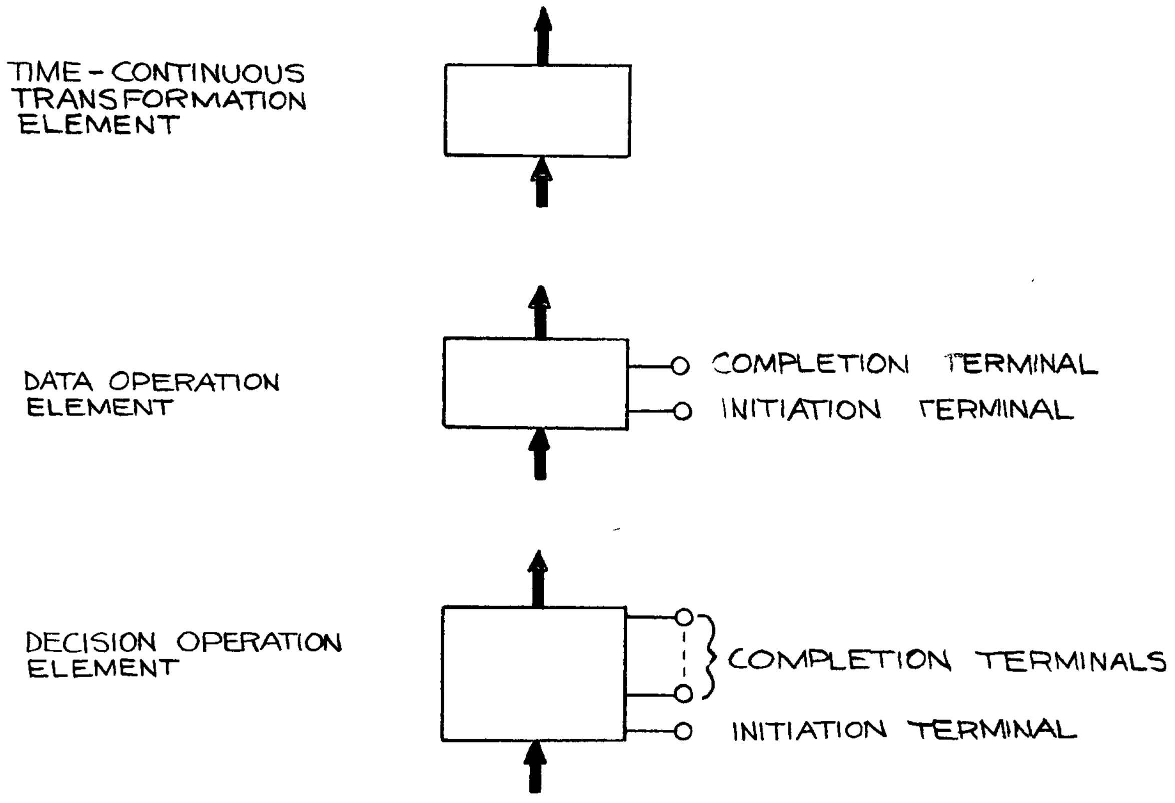


Fig. 2

The order or sequence in which operations are performed is determined entirely by the structure of the Sequencing Network. This network is composed of signal nodes, calling elements, and interconnecting pathways. A <u>signal node</u> is an element which provides for the merging or branching of control signals. There are several types, two of which are shown in Fig. 3. A <u>calling element</u> is one which, when activated by a control signal on its initiation terminal, causes an operation to take place and, when signaled of its completion, produces its own completion signal in turn. An <u>operation node</u> is a calling element for data operations, and a <u>decision node</u> is a calling element for decision operations.

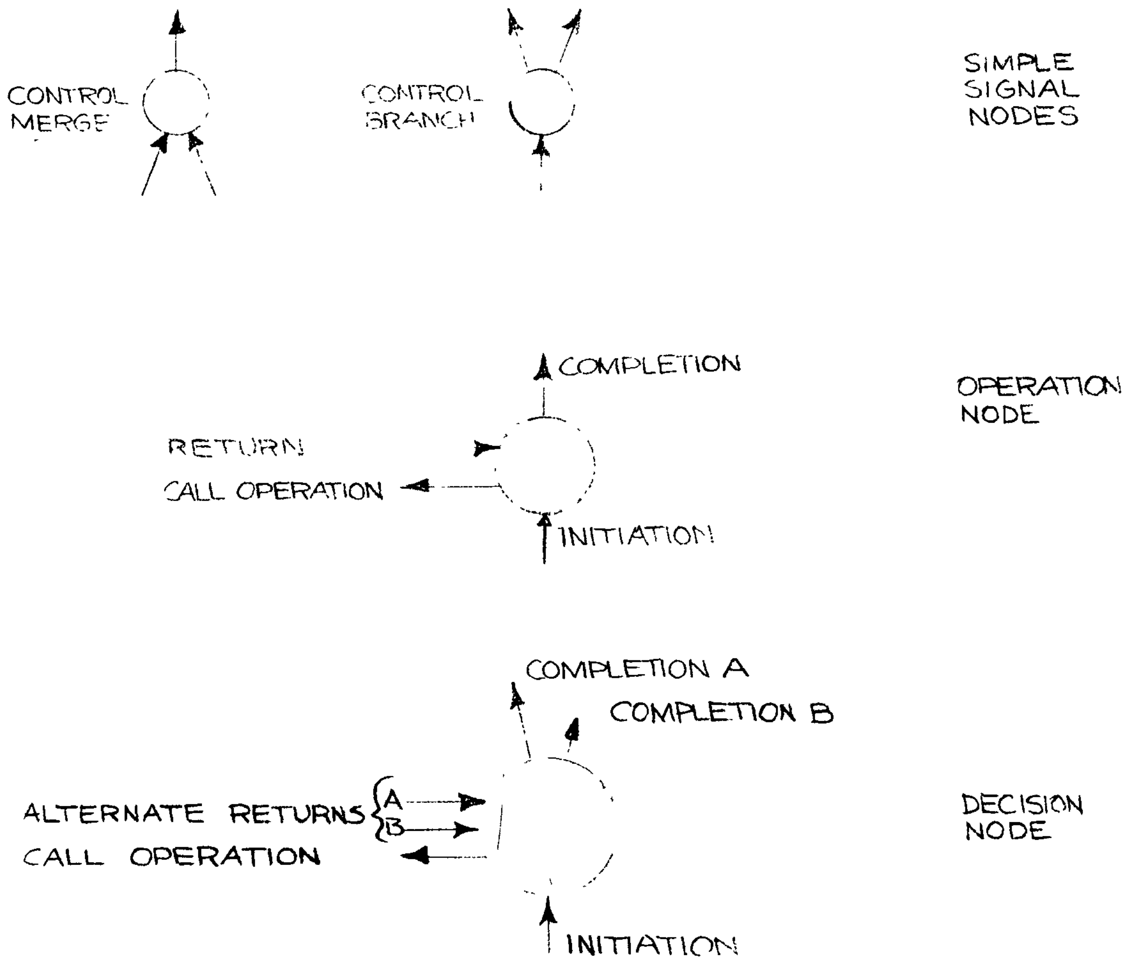


Fig. 3

Control within a macromodular system is asynchronous, that is, each event in a sequence of events is initiated by the completion signal from the preceding event. The simplest way of arranging for this would be to connect the completion terminal associated with each operation to the initiation terminal associated with the next operation. This scheme, though simple and effective, does have the limitation, however, that once the control terminals for an operation have been connected for one sequence, it is no longer possible to incorporate the operation into any other sequence. We therefore revise the scheme as follows: rather than connect to the terminals associated with the operation, we connect instead to the terminals of a calling element associated with the operation, as shown in figure 4. Since any number of calling elements may call the same operation, an operation may now occur in as many sequences as necessary. Figure 4 illustrates this for two different sequences, sequence z, y, x and sequence z, x. Since both sequences include the operations z and x, they initiate the operations via calling elements. Calling elements are not needed for operation y, however, as it appears in only one sequence and can therefore be incorporated by connections directly to its control terminals.

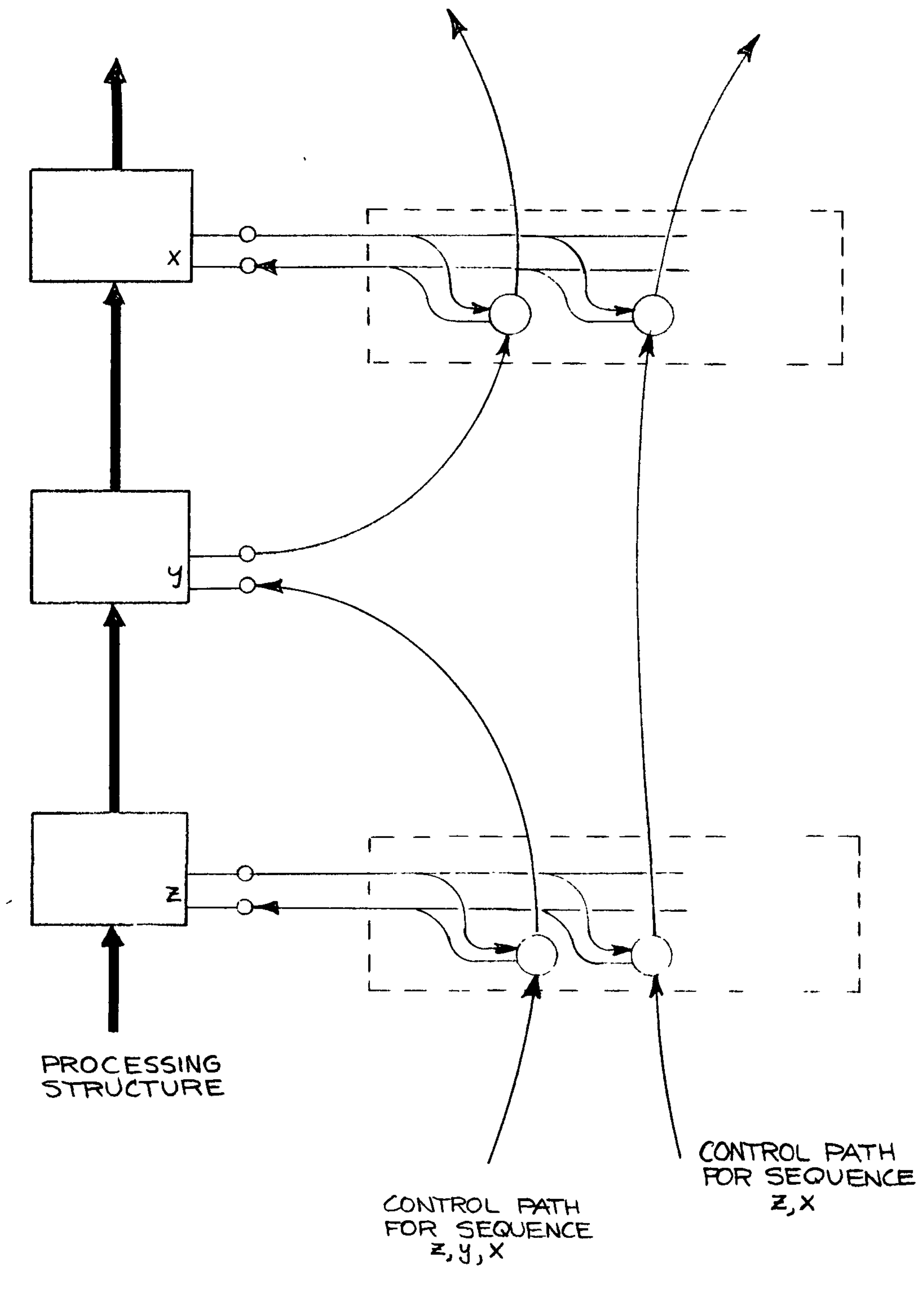

x
y
z
PROCESSING STRUCTURE
CONTROL PATH FOR SEQUENCE z, y, x
CONTROL PATH FOR SEQUENCE z, x

Fig. 4

The control structure described above has the additional feature that an entire sub-sequence of operations can be called by a calling element (Fig. 5). This allows sequences to share common sub-sequences, thereby reducing the number of control elements and interconnections required in a system.

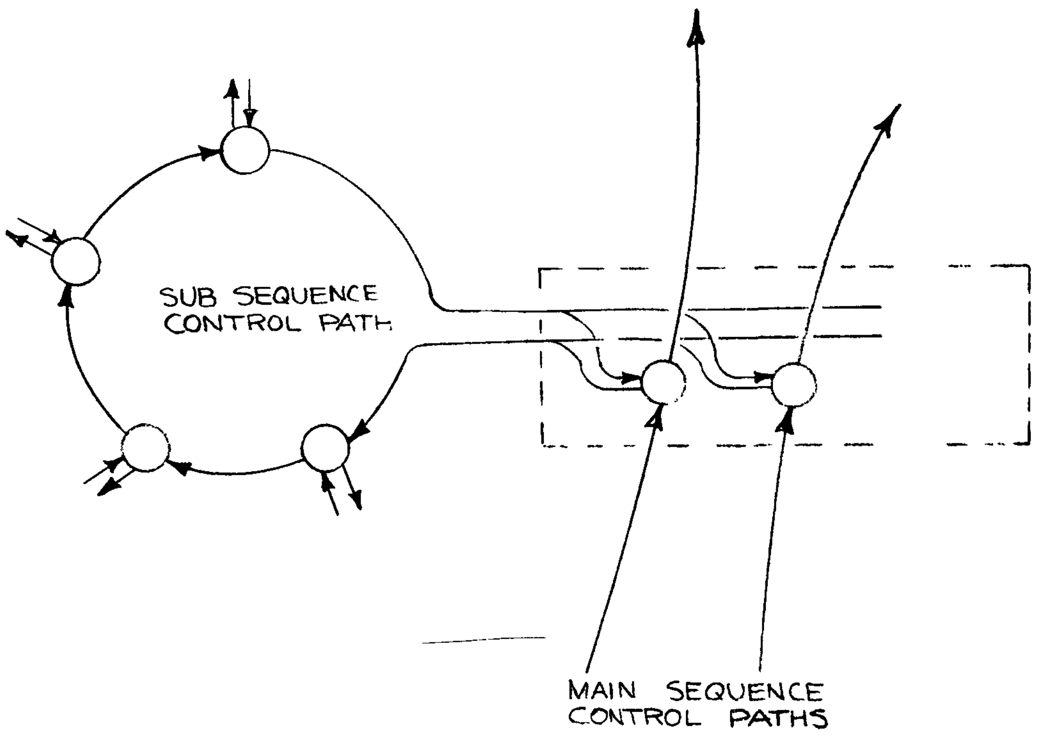


Fig. 5

The control elements and interconnections that define a given sequence are said to be the control path for that sequence. The Sequencing Network is therefore the control path for the entire system.

Data Validation

In order to perform an operation on data or make a decision based on the value of data, it must first be established that the data is available at the point at which it is to be used. There are two general requirements to be met:

1) The data source must present the data in stable form, that is, it must have completed any operation initiated earlier which may affect the data. This requirement stems from the fact that, in the interest of speed, a data element generates a completion signal as soon as it no longer relies upon its input data, regardless of whether or not its operation is complete and its outputs stabilized.
2) Enough time must be allowed for the stable data signals to propagate to the point of use regardless of the length of the pathway.

When data is used in the immediate locality of its source, i.e., within the same data processing element, allowances for stabilization and propagation times are made by the element itself. When the source is remote from the point of use, a procedure known as Data Validation (Fig. 6) is used to guarantee that the requirements are met.

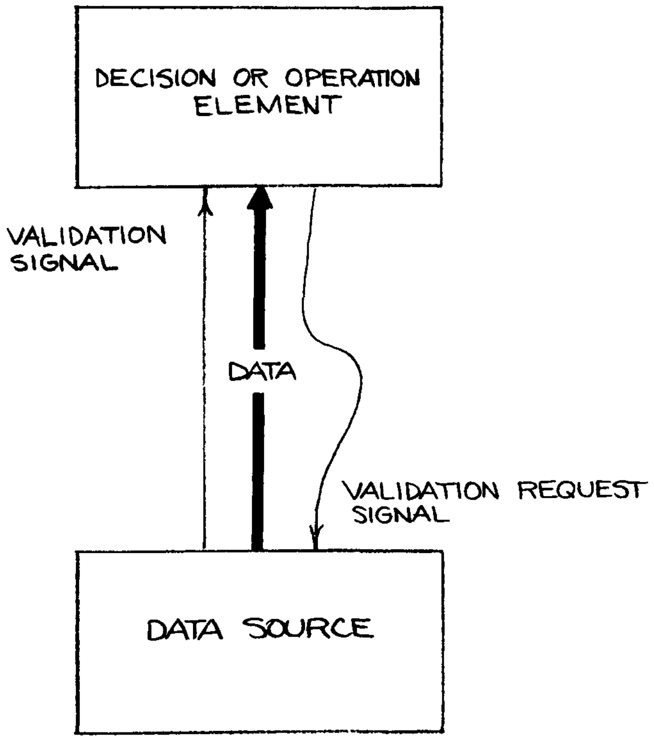


Fig. 6

In this process, before data from a remote source is used, a Validation Request Signal is transmitted to the module containing the data. In response, that module generates a Validation Signal as soon as the signals on its data output terminals are stable. This signal travels back to the point at which the data is to be used along a pathway exactly parallel* to the data pathway, thereby providing time for data propagation along that pathway regardless of its length. The signal is made to travel somewhat slower than the data so that when it arrives at the point of use, the data is guaranteed to meet both of the above requirements and can be used immediately.

* The data Validation Signal travels on a special channel in the same cable that carries the data thus insuring that its path is exactly parallel to that traveled by the data. For simplicity, the Validation Request Signal also travels on another channel of the data cable.

THE MACROMODULES

We now proceed to give a functional description of the individual macromodules and illustrate their roles in various systems. Processing Network elements are introduced first, and this is followed by a discussion of the various Sequencing Network elements. Power connections are omitted from all figures to avoid obscuring the logical point being illustrated.

Cables

Data paths are constructed with Data Cables, control paths with Control Cables. These cables are made in a limited number of lengths, but cables of any length can be formed by using signal-standardizing extender units. A Control Cable contains a single channel for transmission of a control signal. A Data Cable contains 14 channels, 12 for the transmission of data and one each for the Validation Request Signal and the Validation Signal. In the illustrations, Data Cables are drawn with heavy lines and Control Cables are drawn with thin lines.

Registers

The basic Register Module (Fig. 7) contains a 12-bit register together with logic for the operations clear, complement, shift left, shift right, and count (index) Control terminals for each operation are mounted on one side of the module, and connectors on the bottom and top provide for input and output of data.

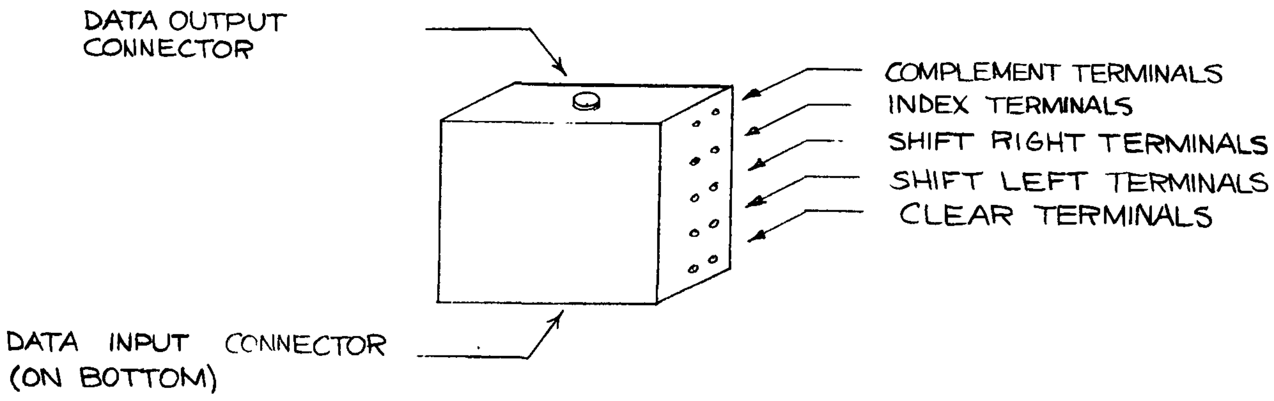


Fig. 7

Registers of any length can be formed by plugging these modules together, lateral pathways being joined by interface connectors (not shown in the figures). Figure 8 shows a 36-bit register formed from three Register Modules. Plugging the modules together obscures the control terminals on all but the rightmost module so that the resulting register has but one set of control terminals, and these provide for the control of the whole register. This is particularly important as it makes control of an operation independent of register size. When modules are plugged together, special circuits within each module are coupled in such a way as to compensate automatically for the increased lateral signal propagation times. Proper operation is thereby guaranteed regardless of register length.

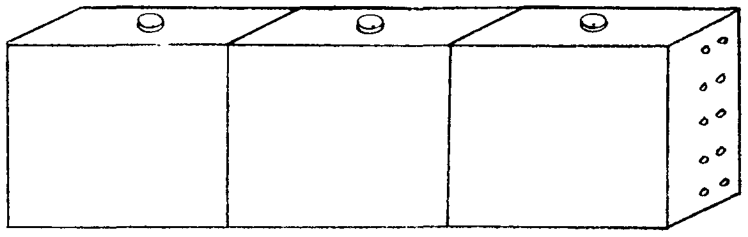


Fig. 8

Transfer Operations

Data transfers from one Register Module to another require the use of a Data Gate module. This unit plugs directly into the base of the receiving Register Module and is connected by means of a Data Cable to the output of the data source Register Module (Fig. 9). Twelve bits are transferred in parallel, and the transfer initiation and completion terminals appear on one side of the module. Transfers are copying operations and as such do not alter the information at the source.

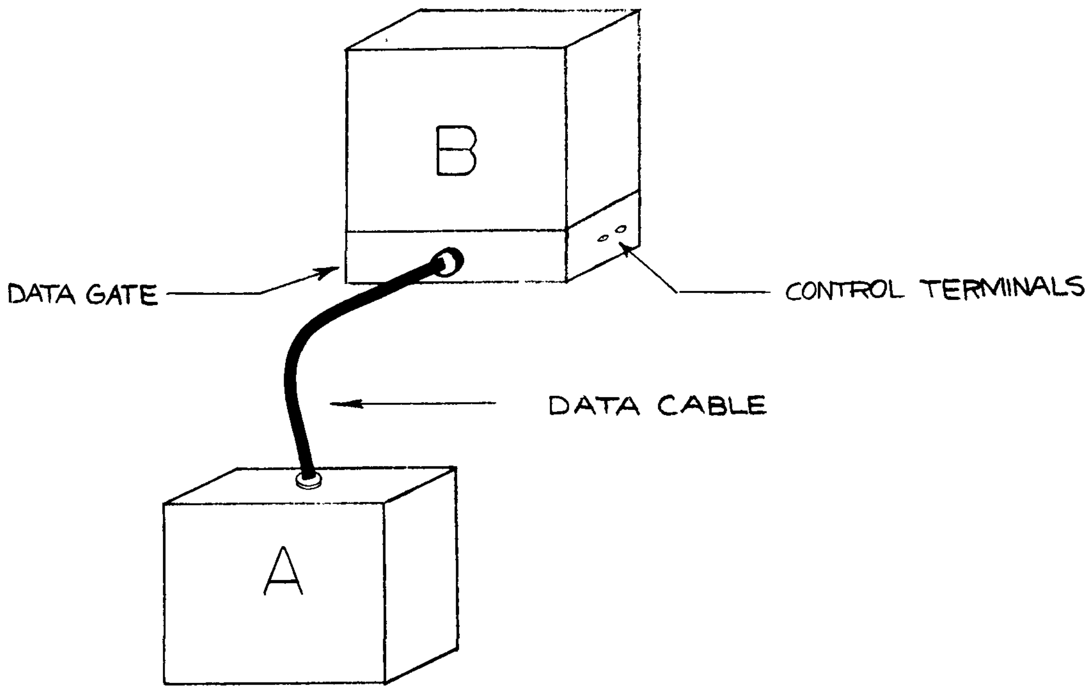


Fig. 9

A separate Data Gate is required for every transfer path into a Register Module. If, therefore, a Register Module is to receive input from n other Register Modules, n Data Gates are required. Figure 10 shows a Register Module equipped with transfer paths from three other Modules. Note that only one of the Data Gates plugs into the Register Module itself; each of the others plugs into the base of another Data Gate. Stacking the units in this way allows each Data Gate to communicate with the receiving Register Module. Any number of transfer paths into a Register Module can be provided by stacking an appropriate number of Data Gates in this manner.

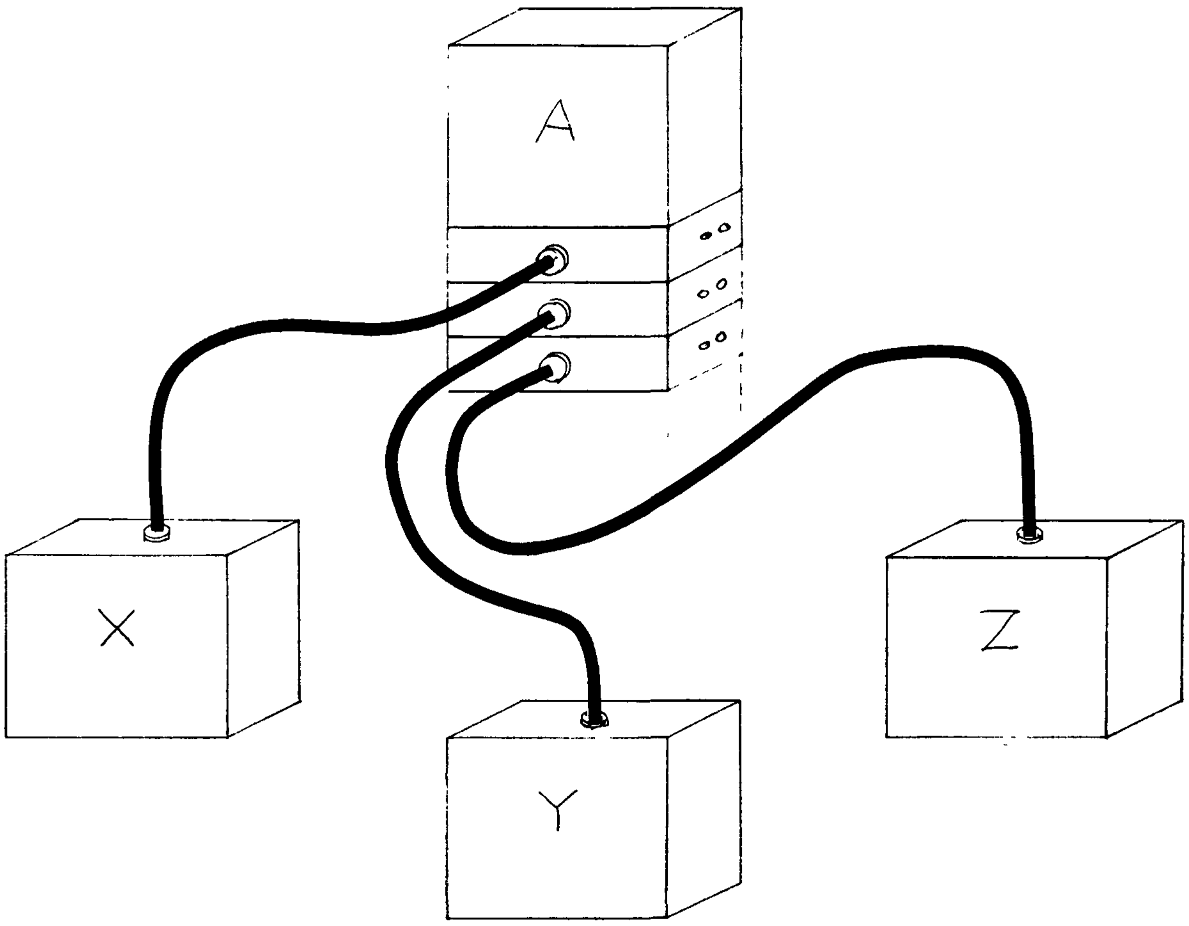


Fig. 10

To transfer information, data must be gated onto bus lines passing through the Data Gates which go up into the Register Module. It is important to consider the action of a Data Gate somewhere in the middle of a stack of Data Gates attached to a Register Module. When a particular Data Gate receives a control signal at its initiation terminal, it performs a data validity check as described previously. This establishes the validity of the data signals at the Data Gate itself. Because of stacking, however, the Register Module must be considered to be an arbitrary distance, and therefore time, away from the Data Gate. Hence, after the Validation Signal arrives at the Data Gate, it is transmitted up into the Register Module along a pathway parallel to but slower than that travelled by the data to the Module. It thus arrives at the Register Module slightly later than the gated data which can therefore immediately be transferred into the register. A completion signal, indicating that the transfer has been accomplished, is then returned down through the stack to exit at the active Data Gate's completion terminal.

Data Gates can be plugged together to provide for transfers into longer registers. The resulting combination has but one set of control terminals for the control of the whole gate. Figure 11 shows a 24-bit register with transfer paths from another 24-bit register.

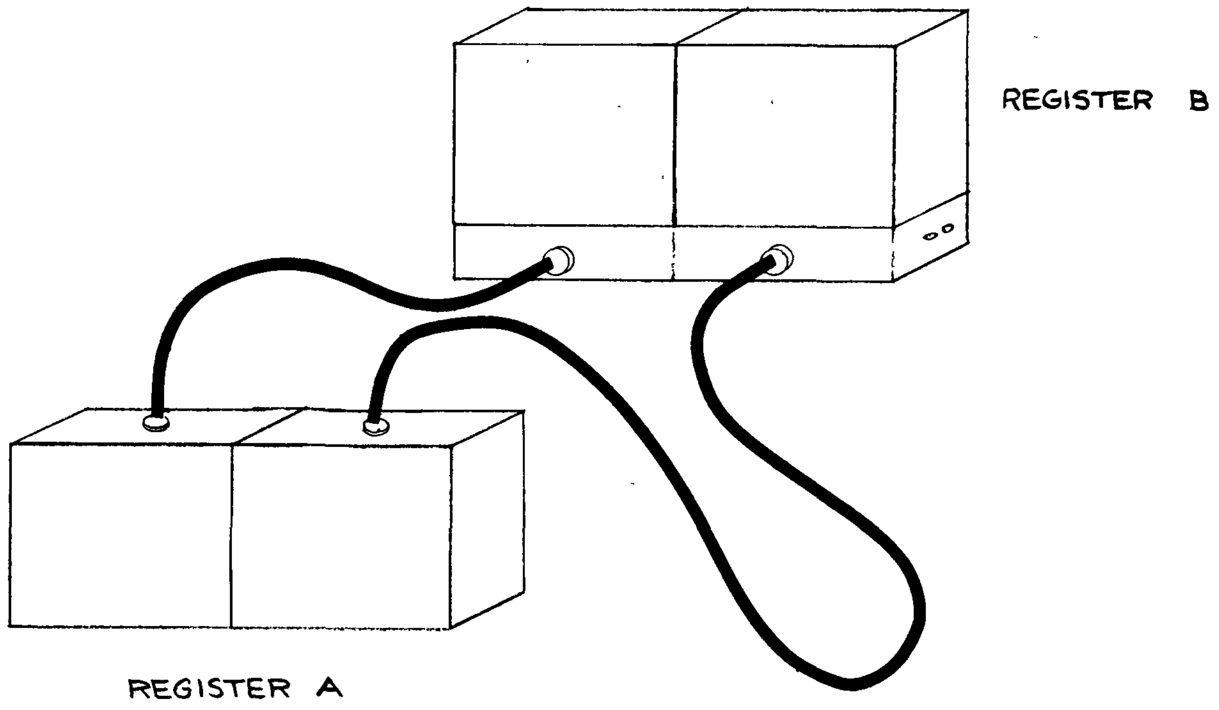


Fig. 11

There is no requirement that the interconnecting Data Cables for the register segments be of identical lengths To compensate for differences in length, the validity of the data from each source module is individually checked via its own Data Cable.

In order to permit the transfer of information from a single source into more than one destination module, a Data Branch unit is used (Fig. 12).

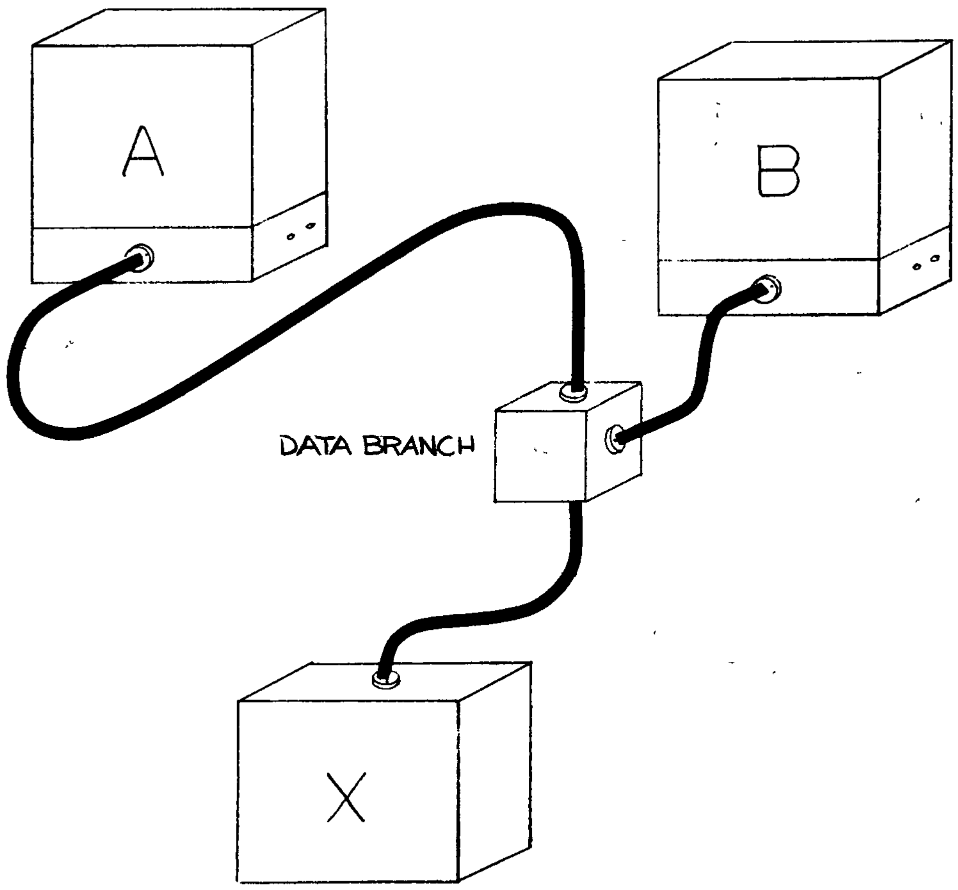


Fig. 12

Data Branch units may be interconnected (see Fig. 13) to provide any number of connections to the same source.

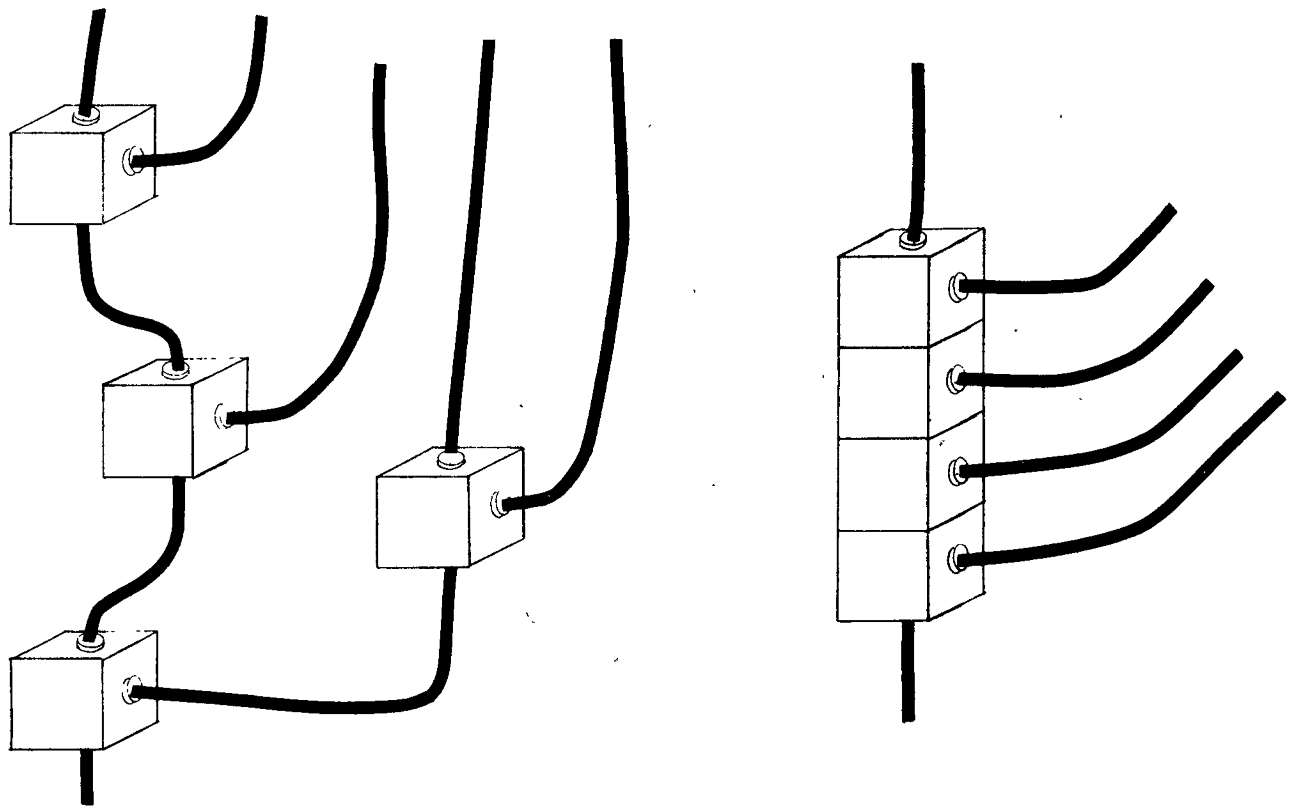


Fig. 13

The Data Branch unit is only a way-station and not a source of data. Therefore, when it receives a Validation Request Signal, it relays it down-stream to the source. The unit remembers which branch carried the Validation Request Signal, and when the Validation Signal returns, it is relayed up that, and only that, branch. The importance of this feature is made clear in Fig. 14.

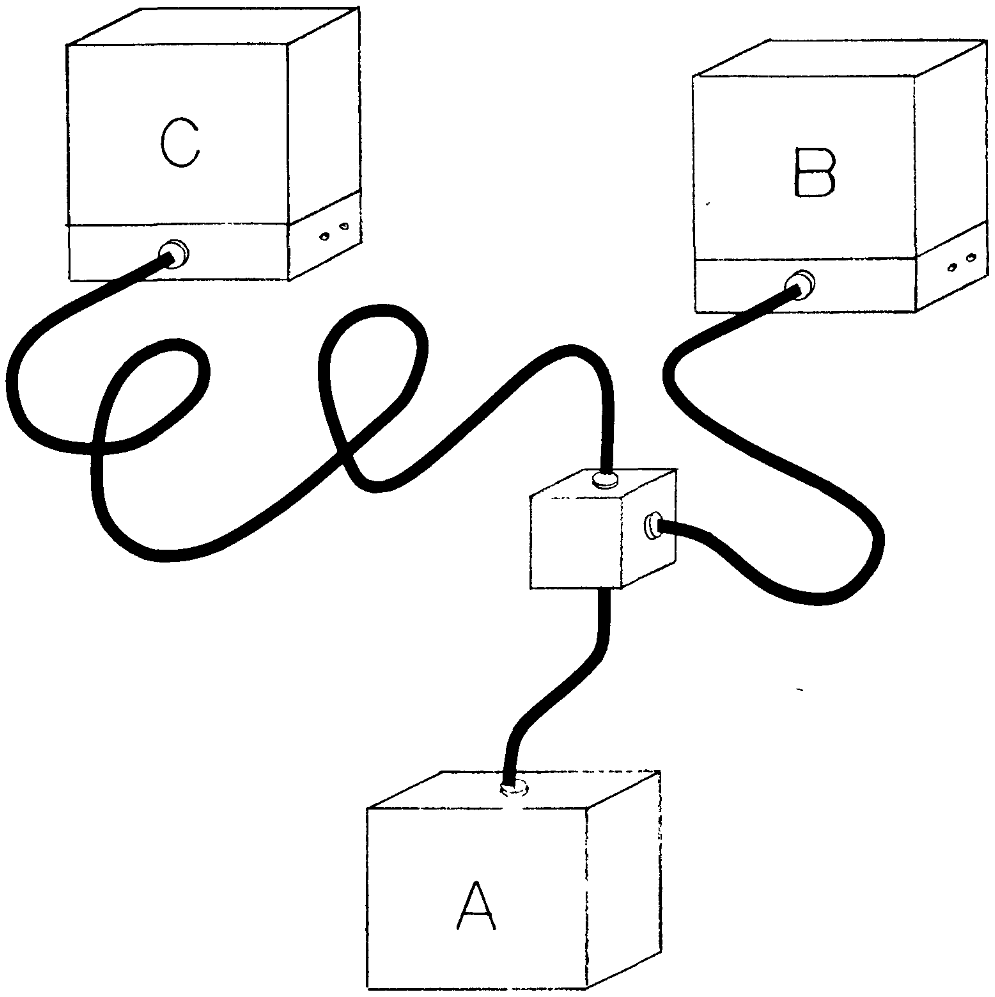


Fig. 14

Suppose that the sequence

1) A $\rightarrow$ B
2) COMPLEMENT A
3) A $\rightarrow$ C

is to be performed, and that Validation Signals are transmitted from the Data Branch unit toward both B and C. It would then be possible for the Validation Signal from step 1 to arrive at the A $\rightarrow$ C Data Gate after the Validation Request Signal for step 3 had been sent but before the corresponding Validation Signal for step 3 returns. At that time, the original uncomplemented data from A would still be present at the Data Gate and would be transferred into C by the premature arrival of the Validation Signal from step 1, and an erroneous completion signal would be produced. Such incorrect behavior is prevented by arranging matters so that a Validation Signal is returned only to the element which issued the Validation Request Signal.

The Data Branch unit embodies all of the features essential to a Data Cable extender (i.e., data signal amplification, ability to relay data validation signals, etc.). This unit therefore doubles as an extender unit for Data Cables (Fig. 15).

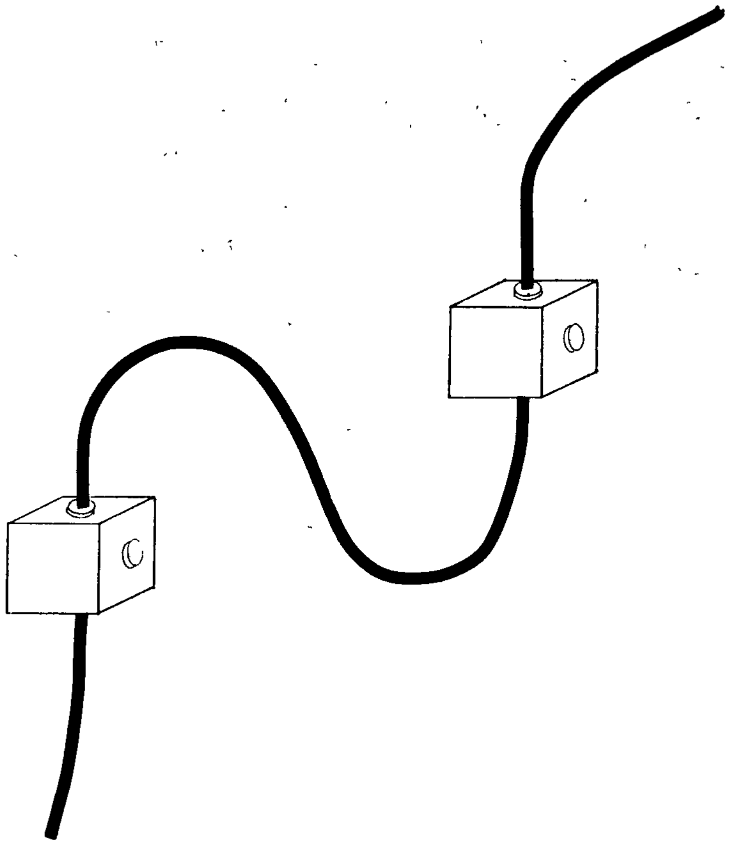


Fig. 15

Finally, it must be pointed out that two registers cannot exchange information without the aid of a third register. This follows from the fact that simultaneity of events in different parts of an asynchronous system cannot be assumed.

Memory

The Memory Module has a capacity of 4096 12-bit words and contains, in addition to a memory array, all required drivers, addressing logic, sense amplifiers, internal address and data registers, etc. The units may be plugged together, as discussed below, to form larger memory systems. (Inter-module connectors are omitted from the figures for simplicity.)

Figure 16 shows a simple arrangement in which one Memory Module is used. The unit marked S is a Register Module whose output is connected to the Address Input of the Memory Module, the unit marked C is a Register Module whose output is connected to the Data Input of the Memory Module, and a Data Gate attached to Register Module B is connected to the Data Output of the Memory Module. Initiation and completion terminals are provided for both Read and Write operations.

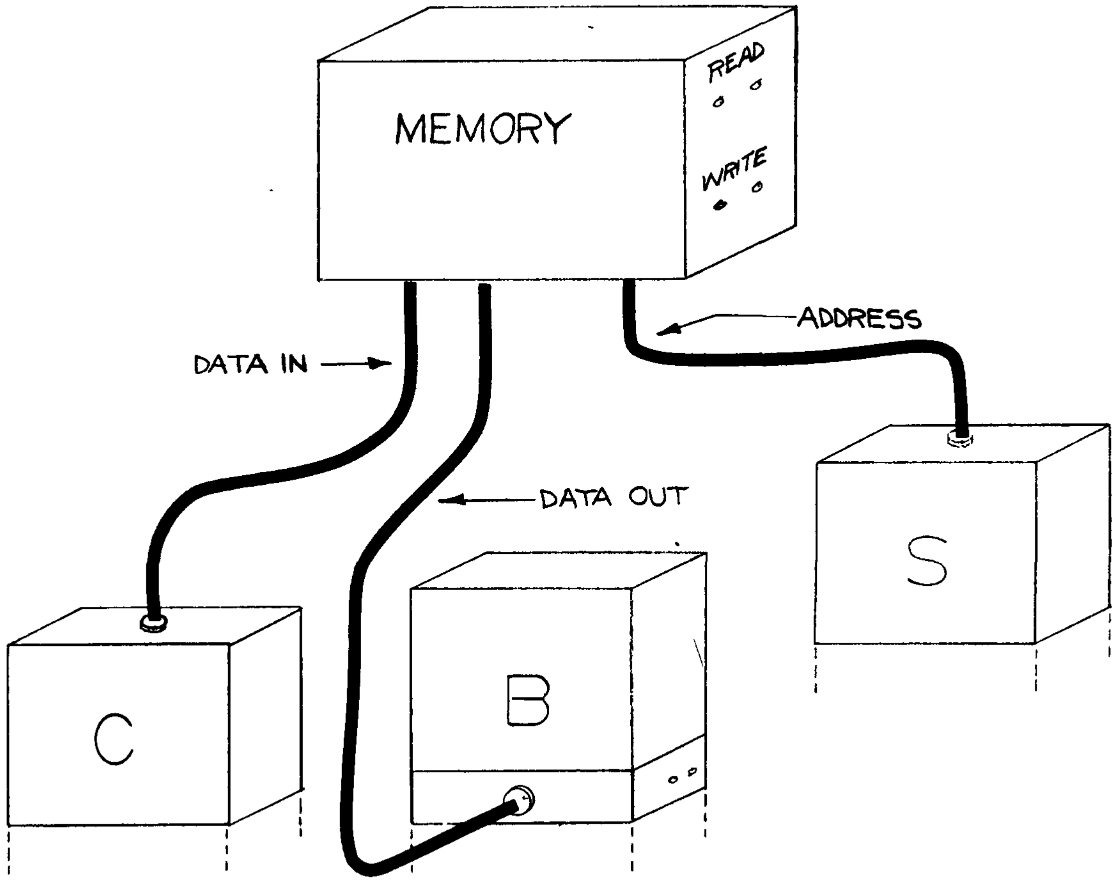


Fig. 16

The process of Reading involves two steps:

Step 1. An address is transferred into the memory's internal address register.

Step 2. The Read operation is performed and the word obtained is presented at the unit's Data Output terminals.

These two steps are initiated by a control signal on the Read operation's initiation terminal and proceed automatically to conclusion with a completion signal appearing as soon as step 1 is finished so that the system may execute other operations while waiting for step 2, typically a relatively long operation, to be completed. S may be changed any time following the completion signal without affecting the Read operation.

In this example the Memory Module Data Output terminals are connected so that the word obtained from the memory array may be transferred into B. The Read operation's completion signal or any subsequent signal may be used to initiate this transfer operation. If the Memory Module has not yet finished step 2, the Validation Signal to the Data Gate is held up until the word is available.

Writing a word into the memory involves the following set of operations:

Step 1. An address is transferred into the memory's internal address register, and the word to be written is transferred into the memory's internal data register.

Step 2. The Write operation is performed.

These operations are initiated when a control signal is presented at the Write operation's initiation terminal and, as in the case of the Read operation, a completion signal appears as soon as step 1 above is completed. Thereafter, registers C and S may be disturbed without affecting the Write operation.

If an attempt is made to read from or write into the memory while it is still responding to some earlier control signal, the new control signal will be held up until it can be accommodated.

Memory Modules can be plugged together laterally to increase word length, thereby forming what will be referred to as tiers. These tiers may be stacked vertically to increase the number of words. Figure 17, for example, shows a memory system containing 8192 thirty-six bit words.

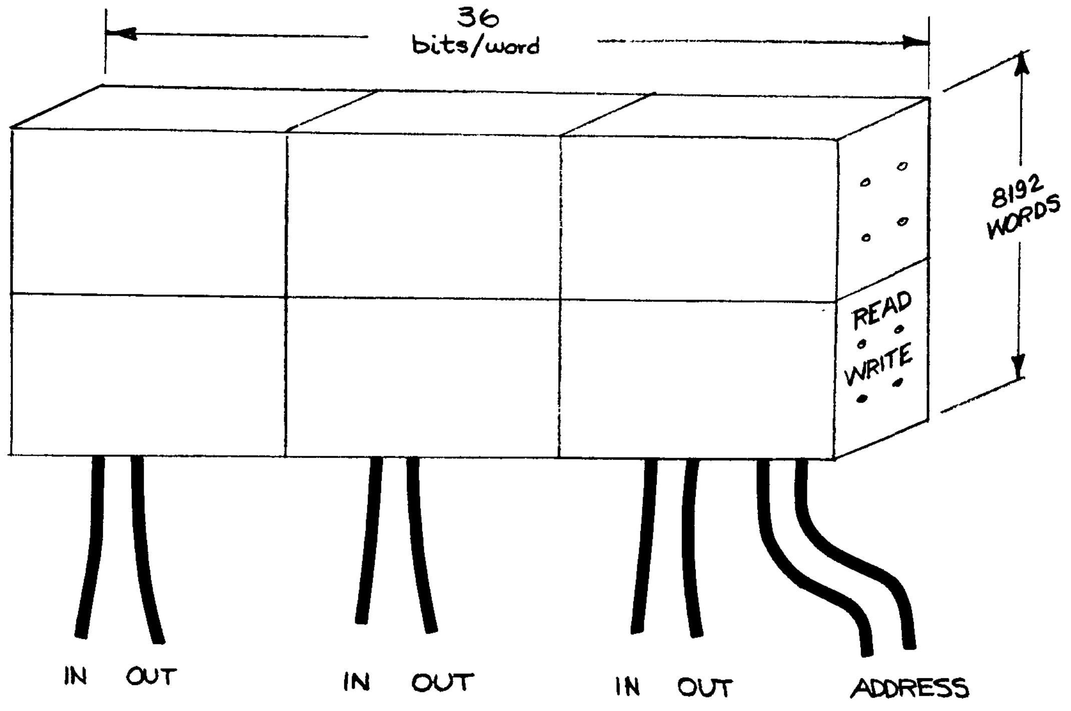


Fig. 17

To permit referencing in memory systems containing more than 4096 words, connectors are provided for additional Address Inputs for selection of the appropriate tier. All address information is provided to the lower rightmost unit which distributes this information throughout the assemblage for proper word selection. Similarly, the entire assemblage is controlled by signals at the Read and Write control terminals on the lower rightmost unit. For input and output data paths, each vertical column uses the data terminals on the lowest tier.

Concurrent operation of different memory tiers is possible. That is, during step 2 of a Read or Write operation in one tier, a Read or Write operation in another tier may be initiated and will proceed immediately. However, the data resulting from one Read operation must be used before another Read operation is initiated.

Time-Continuous Transformations

Three units are provided which perform time-continuous transformations of data. These units continuously present at their outputs a transformation of the data presented at their inputs. When the input information changes, the outputs will reflect this change. These units thus perform their functions continuously in contrast to Register Modules and Data Gates, which react only to command signals presented at a discrete time.

The Junction Unit, Adder Unit, and Function Unit discussed below all perform time-continuous transformations. The outputs of these units may be used in the same way in which Register Module outputs are used.

Junction Unit

A Junction Unit permits one to rearrange bits within a word or to form words from bits selected from several words. The unit has two 12-bit data inputs and a single 12-bit data output. A set of switches on the face of the unit can be set to connect each of the 12 output terminals to any of the 24 input terminals or to supply the value "1" or "0".

Suppose, for example, that we wish to transfer information from parts of two 12-bit registers, A and B, into a third register, C. Specifically, suppose that we wish to copy the rightmost four bits of A into the leftmost four bit positions of C, the leftmost four bits of B into the rightmost four bit positions of C, and set the four middle bits of C to the binary value 1011. To accomplish this we connect the units as shown in Fig. 18, setting the switches on the Junction Unit to perform the transformation as shown in Fig. 19. The word made up by the Junction Unit is transferred into C via a Data Gate.

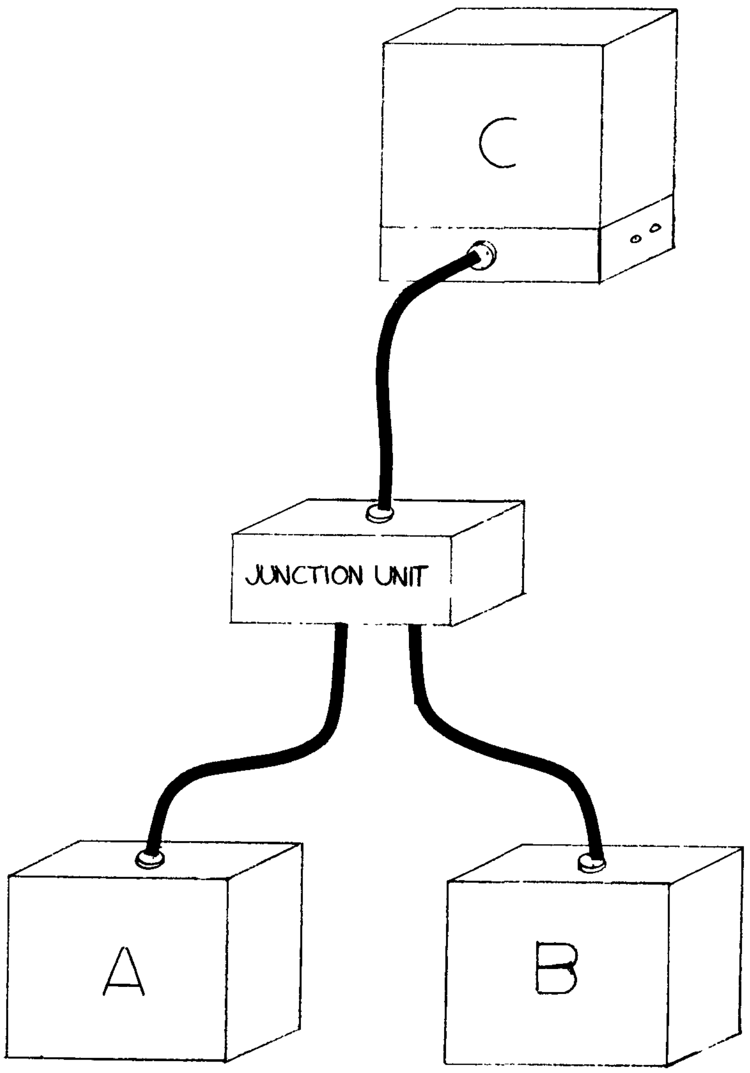


Fig. 18

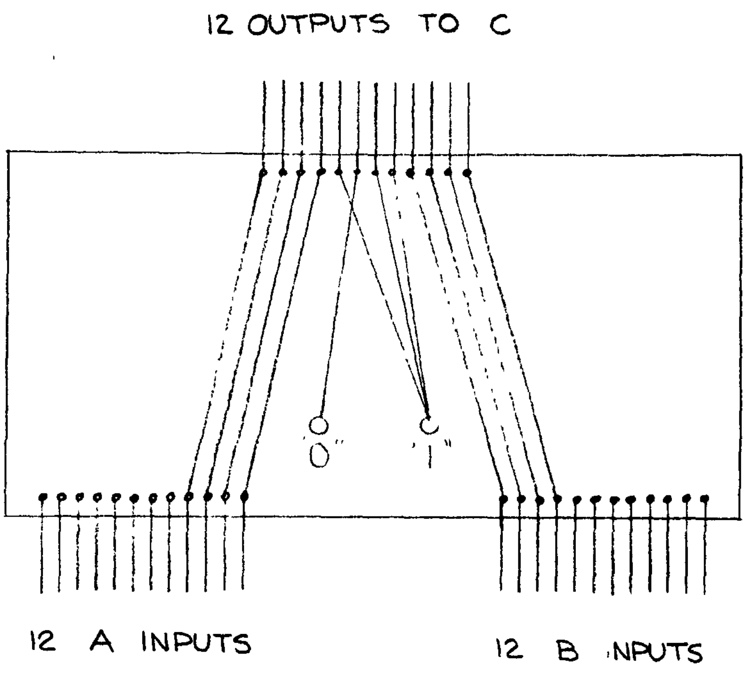


Fig. 19

Let us look briefly at the process of data validation in the preceding example. When the Data Gate is activated, it transmits a Validation Request Signal to the Junction Unit which in turn sends Validation Request Signals to each of the sources, A and B. When Validation Signals have returned to the Junction Unit from both sources, the Junction Unit transmits a Validation Signal back to the Data Gate.

For the permutation of bits within a 12-bit word, only one of the two inputs is used. The absence of a cable on the unused input connects the Validation Request Signal line to the Validation Signal line on that side, thereby indicating constant validity.

Adder Unit

The basic Adder Unit is like the Junction Unit in general form. It receives inputs from two 12-bit sources and, from these, forms a sum which is continuously presented at its output terminals. Figure 20 shows an Adder connected to form the sum of the numbers in A and B. An initiation signal at the Data Gate transfers the sum into C.

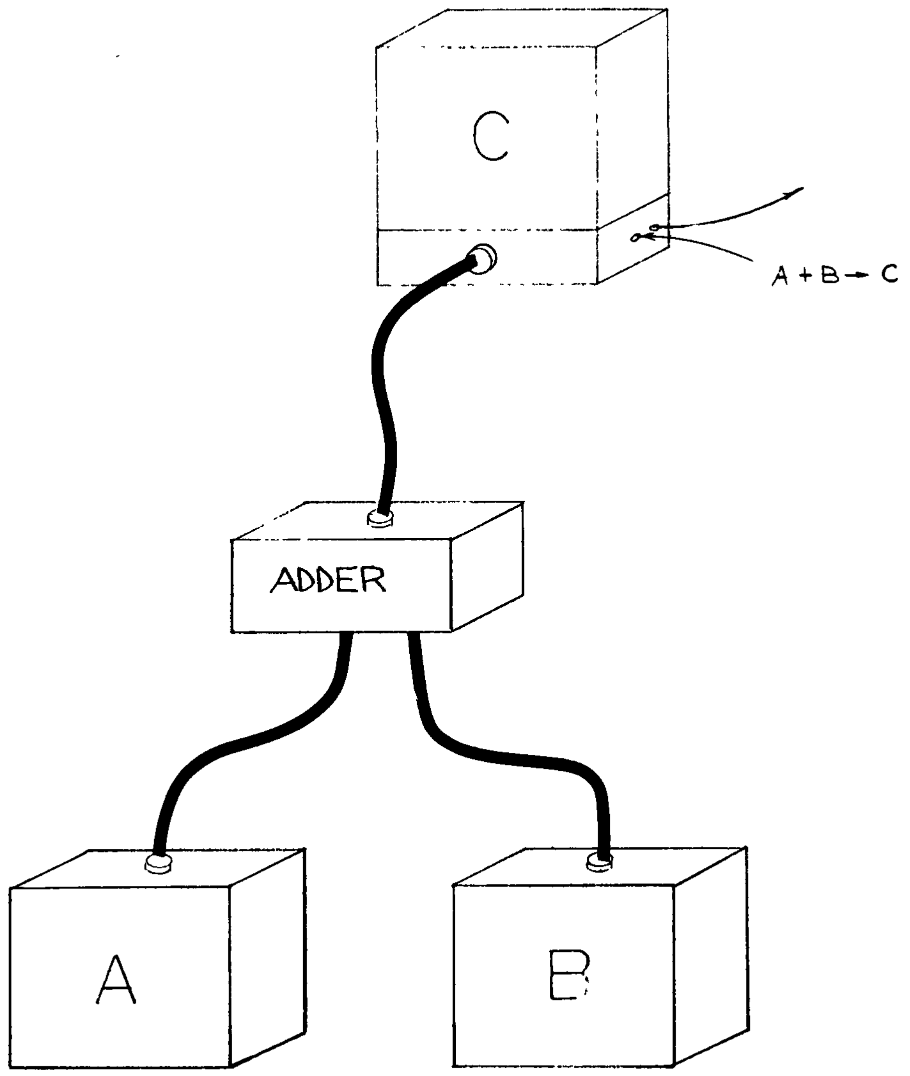


Fig. 20

Adders for larger registers are formed by plugging Adder Units together laterally. Carries are propagated from unit to unit through connectors at their interface, which also interconnect internal circuits compensating for the increase in carry propagation time. Figure 21 shows a 24-bit accumulator, A, to which the contents of register B are added when a control signal is presented at the Data Gate.

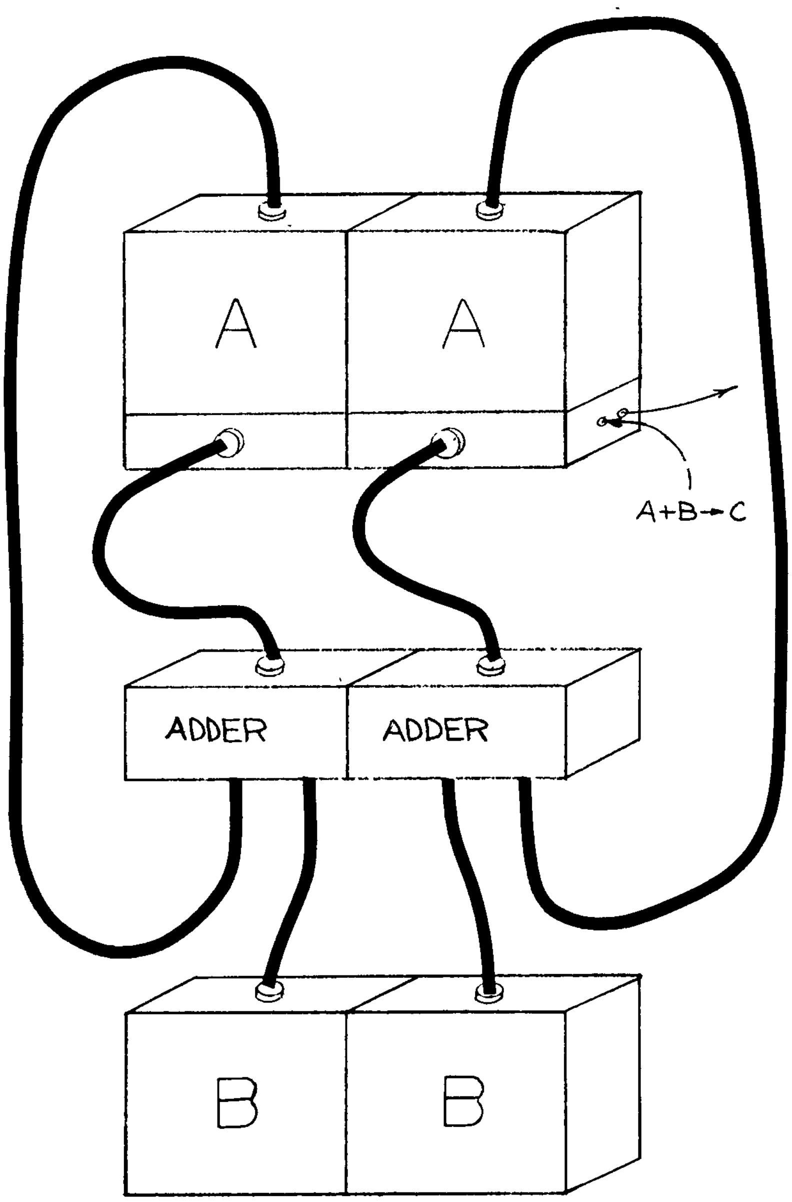


Fig. 21

Before a sum is used at a destination module, a validity check is performed. When an Adder Unit receives a Validation Request Signal, it in turn sends Validation Request Signals to both input data sources. After both sources have returned Validation Signals and formation of the sum has been completed, the Adder Unit returns a Validation Signal to the destination.

The Adder Unit contains three decision nodes which provide for the detection of overflow, negativity, and the numerical sum zero. Their use is discussed below in the section dealing with control decisions.

Function Unit

A Function Unit (Fig. 22) performs three logical operations on a pair of data inputs. These operations are the logical "OR" ($\vee$), the logical "AND" ($\cdot$) and the "EXCLUSIVE OR" ($\circledvee$). The results of these operations are presented at three 12-bit data output connectors.

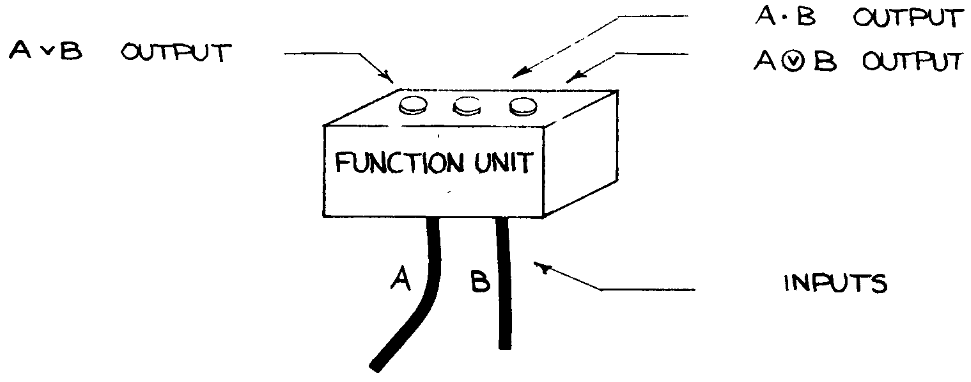


Fig. 22

Data Validation in the Function Unit is similar to that in the Adder and Junction Units. However, as in the Data Branch unit, Validation Signals must be returned only to a destination from which a Validation Request Signal has been received.

Sequencing

In order to perform a desired sequence of operations, control signals are routed along Control Cables from one set of control terminals to the next set in a manner reminiscent of the plugboard programmed machines or the Bell Computer Model VI.[3] Thus to perform the sequence

$$\begin{tikzcd}
{} \arrow[d] \\
\boxed{A \rightarrow B} \arrow[d] \\
\boxed{\text{Index } B} \arrow[d] \\
\boxed{B \rightarrow C} \arrow[d] \\
{}
\end{tikzcd}$$

one would interconnect control terminals as follows.

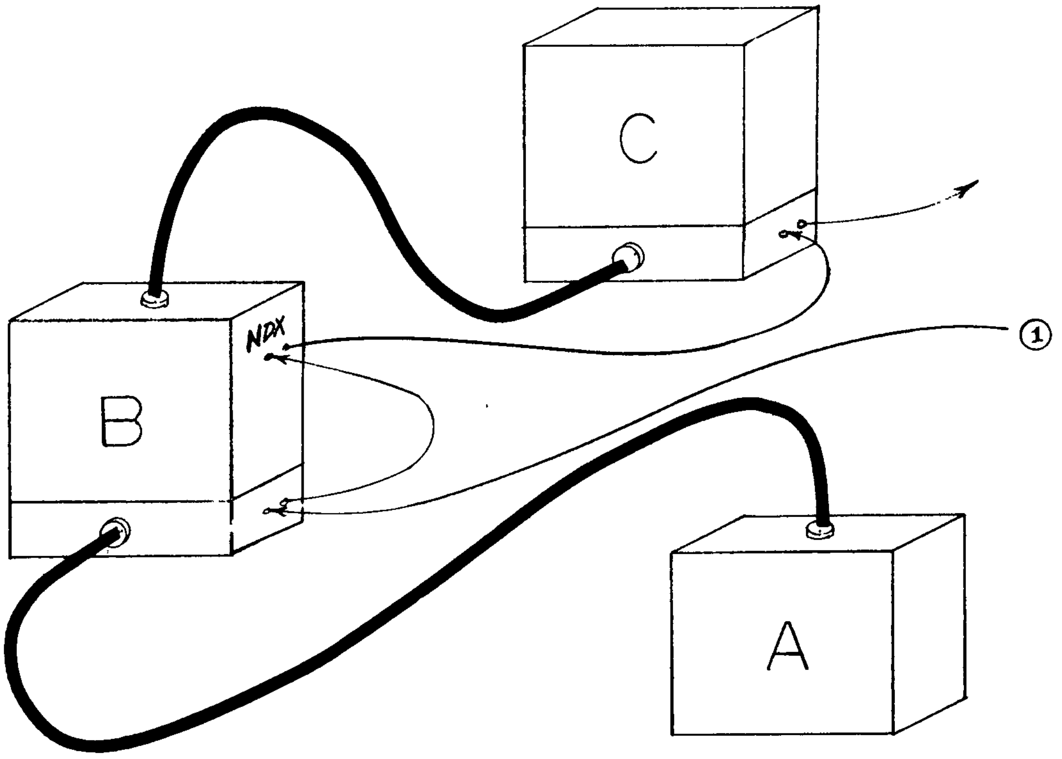


Fig. 23

When a signal on line labeled (1) arrives at the control terminal on the A→B Data Gate, it initiates the transfer of data from A into B. The completion signal from this operation is routed to the control terminal which indexes B, and its completion signal is, in turn, routed to the initiation terminal on the B→C Data Gate, etc.

Concurrent Sequences

Use of the Control Branch unit makes it possible to perform sequences of steps which may be executed concurrently. A control signal presented at the input terminal of such a unit causes control signals to appear on each of two output terminals (Fig. 24).

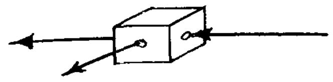


Fig. 24

The Control Branch unit contains amplifiers associated with each output sufficient to drive the maximum standard length of Control Cable, and therefore is may also be used to extend Control Cables. These units may be interconnected to form an arbitrary number of control path branches (Fig. 25).

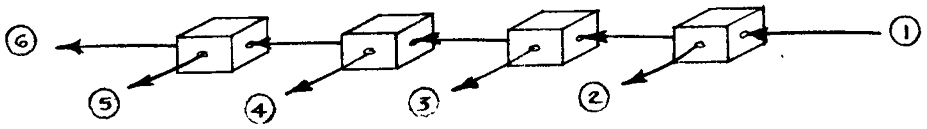


Fig. 25

A signal on line 1 will cause signals to appear on lines 2 through 6. Alternatively, the units may be plugged directly together (Fig. 26).

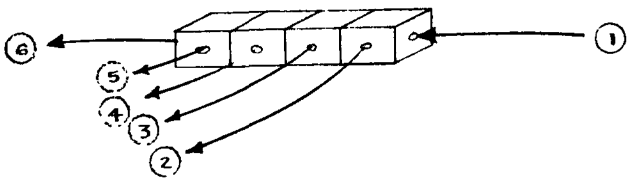


Fig. 26

In general, in the operation of two concurrent sequences, there will be found a point at which ensuing steps can be taken only after all steps of both sequences have been completed up to that point. A Rendezvous Unit (Fig. 27), which produces a signal at its output terminal only after signals have arrived at both of its input terminals, is used at the point of conjunction. (A Rendezvous Unit can thus be identified as a Muller C-unit[4].)

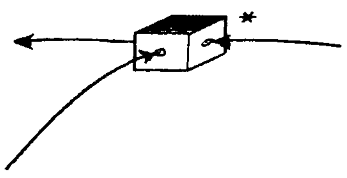


Fig. 27

* The Rendezvous Unit is shown with a darkened top to distinguish it from units of a similar appearance.

For example, suppose that we have a problem which requires several set-up steps, one of which transfers data from register X to register A, and another of which transfers data from register Y to register B. These steps may either be executed sequentially (Fig. 28), or they may be executed concurrently (Fig. 29).

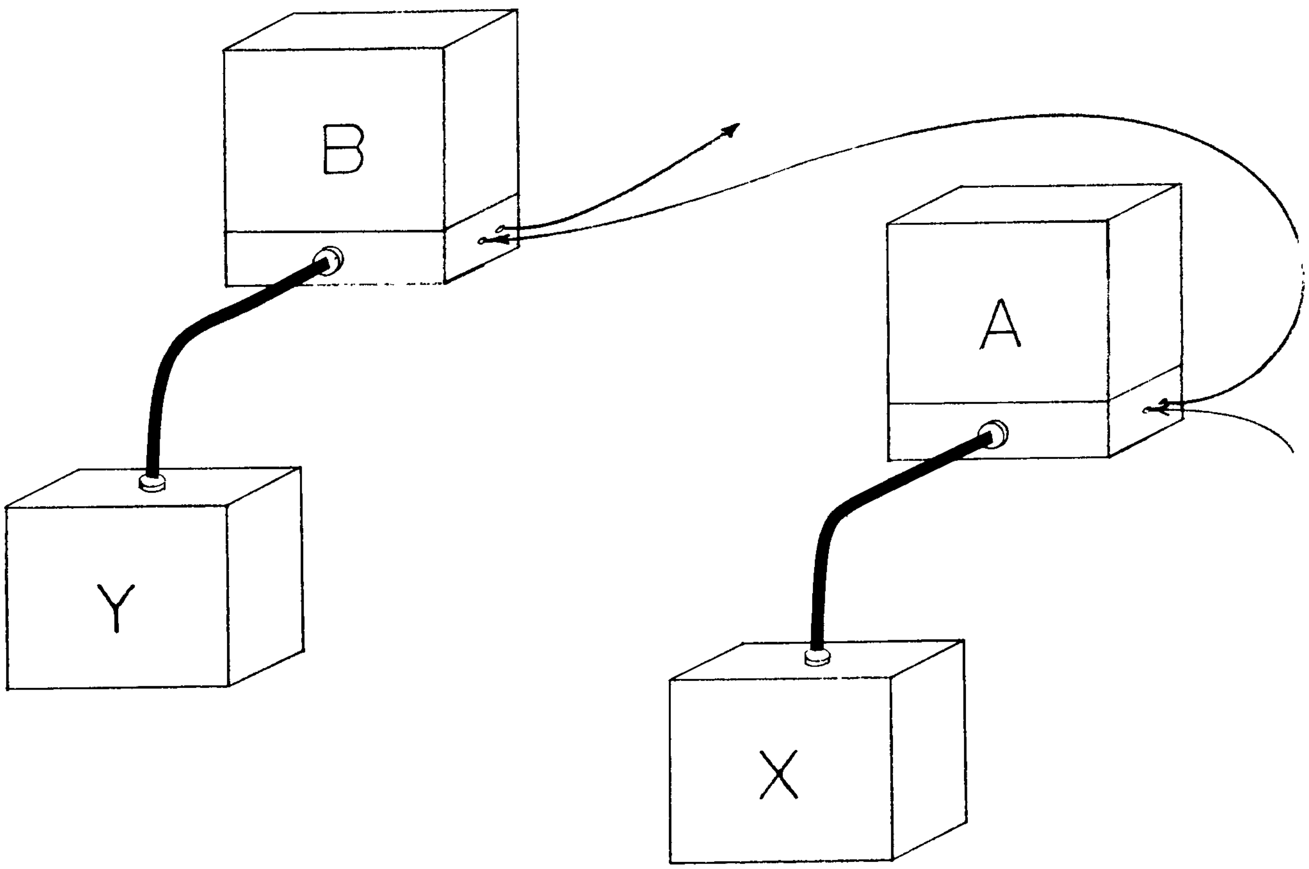


Fig. 28

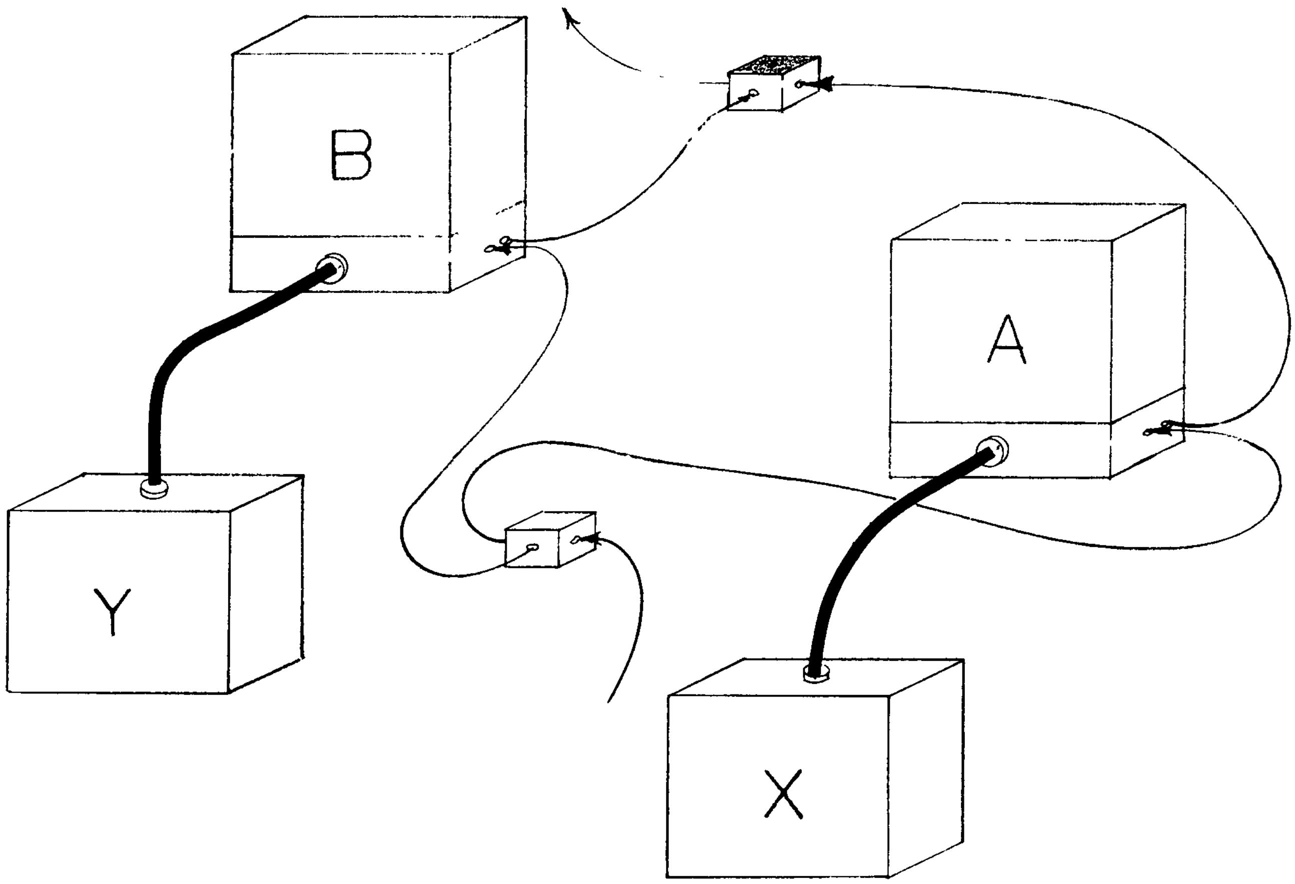


Fig. 29

In the latter case, both transfers are activated and can take place at more or less the same time. As each transfer is completed, a signal is sent to the Rendezvous Unit. When both signals have arrived, the unit sends out a signal which proceeds on to the next step.

A signal indicating the completion of an arbitrary number of concurrent actions can be generated as shown in Fig. 30.

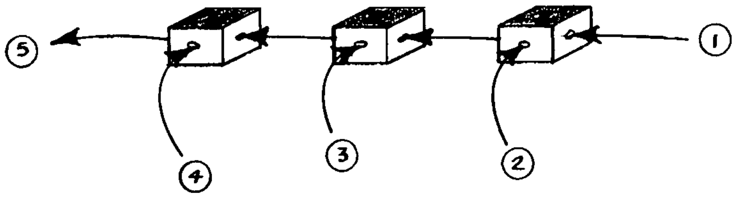


Fig. 30

This signal will appear on line 5 only after signals have been received on lines 1 through 4. The same end may be achieved by plugging the units together as shown in Fig. 31.

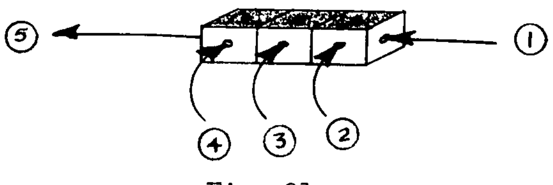


Fig. 31

Call Unit

For those situations in which more than one control path must have access to a single pair of control terminals, calling elements must be used. Five such calling elements are included in a single Call Unit.

A Call Unit is provided with the following terminals:

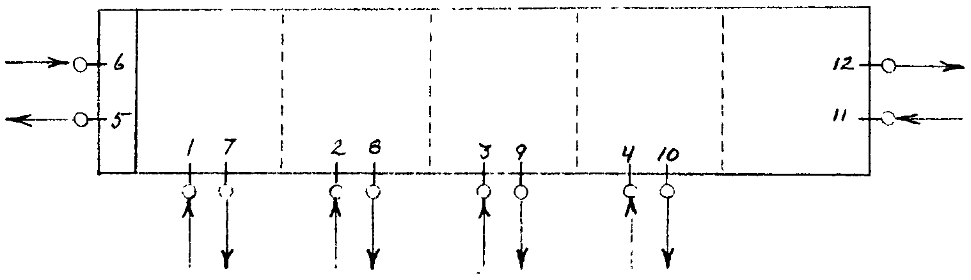


Fig. 32

1, 2, 3, 4 and 11 are input terminals for the elements; 7, 8, 9, 10 and 12 are the output terminals. Terminals of the fifth element are positioned at the right end of the unit to permit easy extension, as discussed below. Dotted lines show the separations between the elements. Whenever a control signal arrives at the input terminal of one of the calling elements, a control signal is presented at terminal 5. When the unit is plugged onto a pair of operation control terminals, terminals 5 and 6 mate with the operation's initiation and completion terminals respectively. The signal produced at terminal 5 thus initiates the operation. When the completion signal from this operation is presented at terminal 6, the Call Unit, in turn, produces a completion signal at the output terminal of the element which called for the operation.

Figure 33 shows a Call Unit attached to the Complement control terminals of a Register Module.

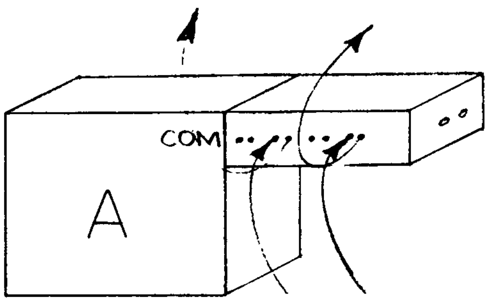


Fig. 33

When Call Units are plugged together (Fig. 34), terminals 5 and 6 of the right hand unit mate with terminals 11 and 12 of the left hand unit. The unit on the right calls its left neighbor via that unit's fifth calling element, the completion signal returning to the right hand unit through the junction of terminals 12 and 6.

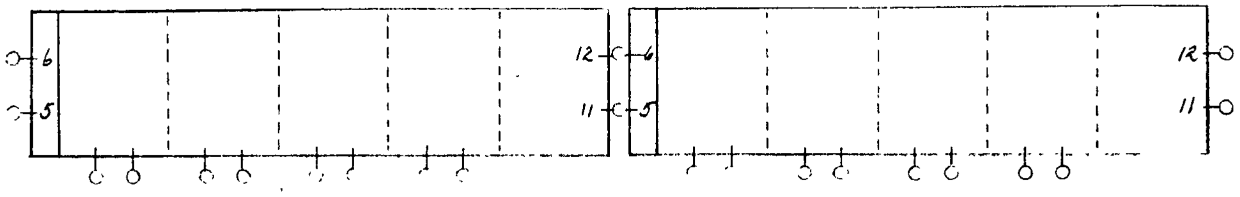


Fig. 34

Figure 35 shows a Register Module with two Call Units on the Index operation terminals.

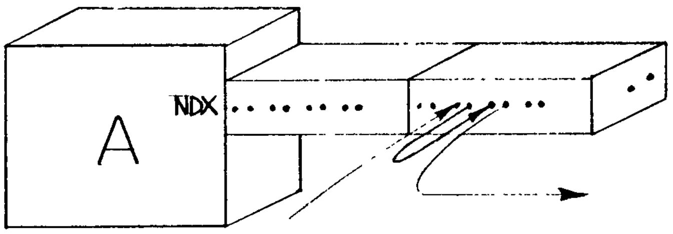


Fig. 35

Note that the control path shown calls for the execution of two Index A operations in sequence.

Call Units may be attached to the control terminals for any operation. Figure 36 shows three control paths, two of which contain a step which transfers A to B, and two of which index A.

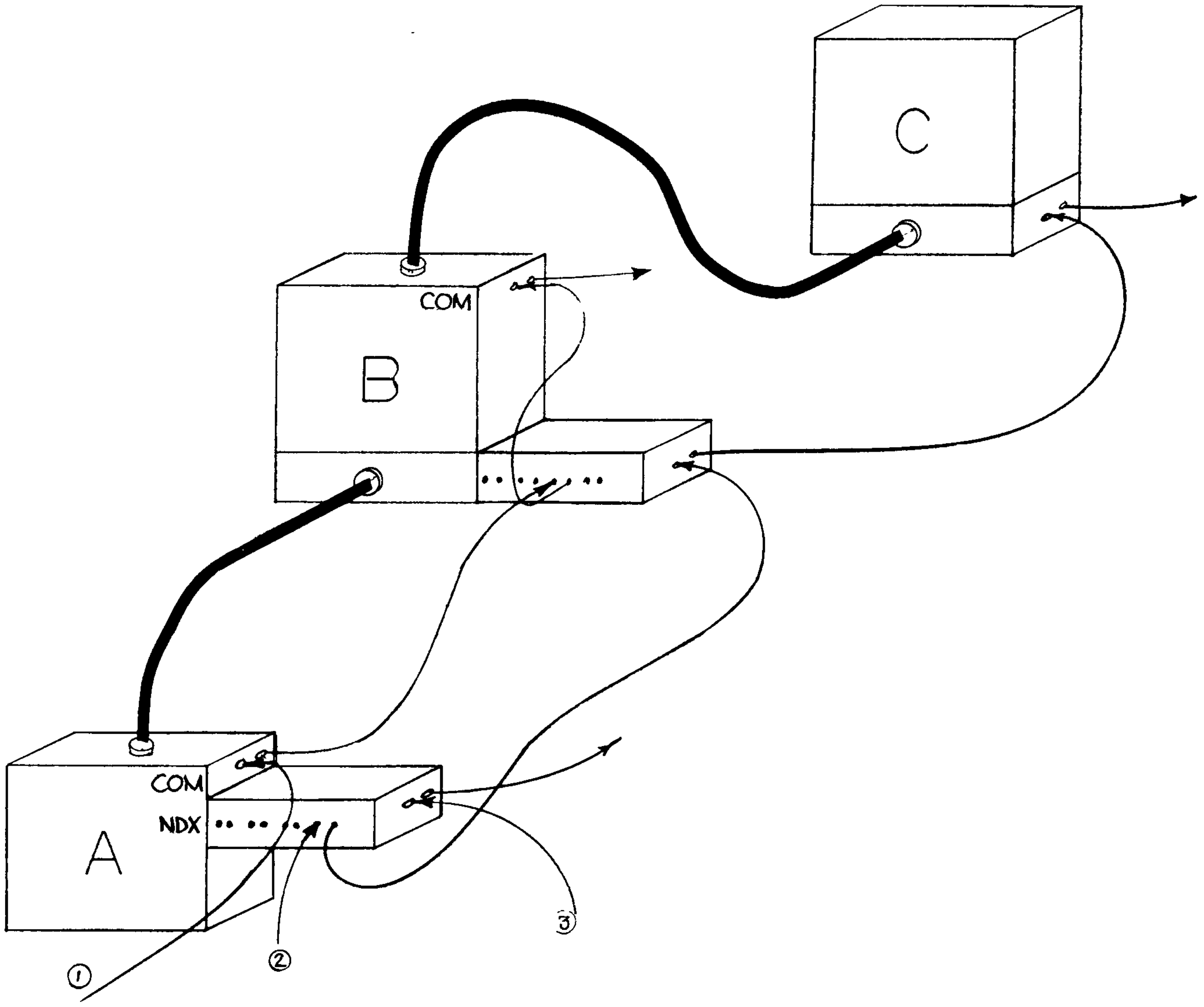


Fig. 36

The three sequences performed are:

(1)	(2)	(3)
COMPLEMENT A	<u>INDEX A</u>	<u>INDEX A</u>
<u>A → B</u>	<u>A → B</u>	
COMPLEMENT B	B → C	

No Call Units are required on the complement A, complement B, and B → C operation terminals, as each of these operations occurs in only one of the above sequences.

A Call Unit need not be plugged directly onto a pair of operation control terminals. It may, instead, be connected by means of control cables (Fig. 37).

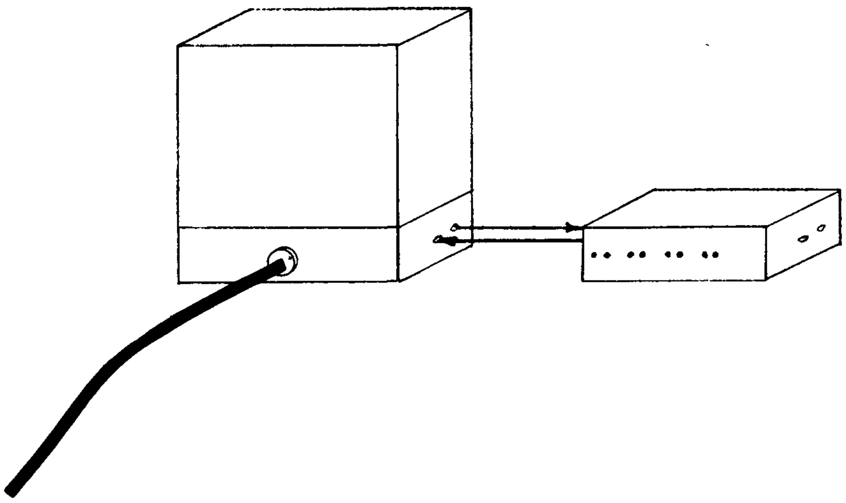


Fig. 37

Similarly, extension may be accomplished with cables (Fig. 38).

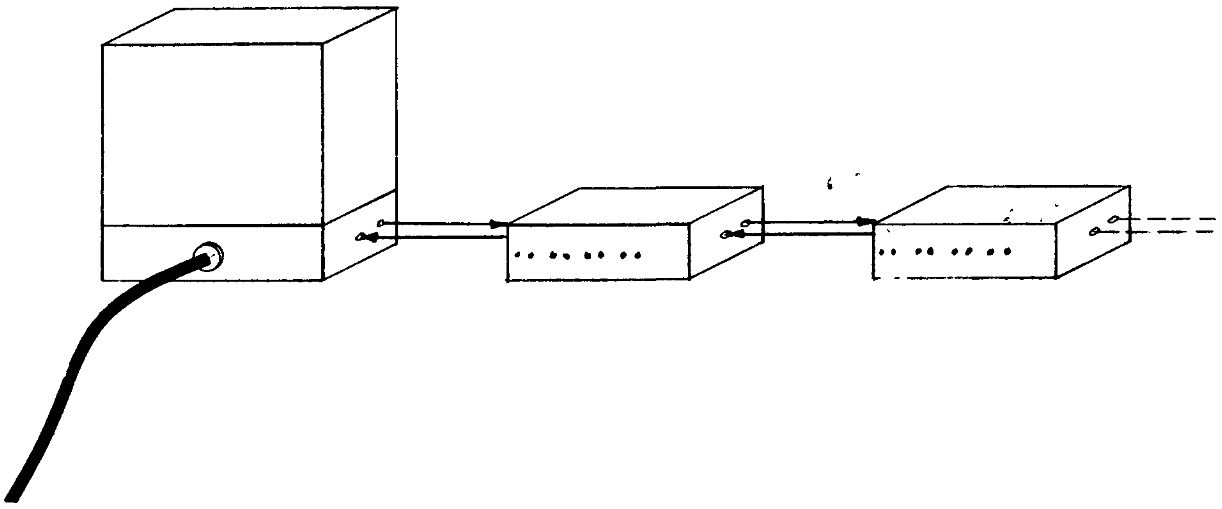


Fig. 38

Sub-Sequence Calling

Suppose, now, that several main sequences have a common sub-sequence (for example, an operand fetch sub-sequence common to several instructions). After completion of the steps of the sub-sequence, each main sequence must continue with its own set of steps (corresponding perhaps to different instruction execution steps).

Figure 39 shows the Call Units which are associated with the steps of the sub-sequence and, without showing the steps explicitly, merely indicates them as S_1, S_2, and S_3.

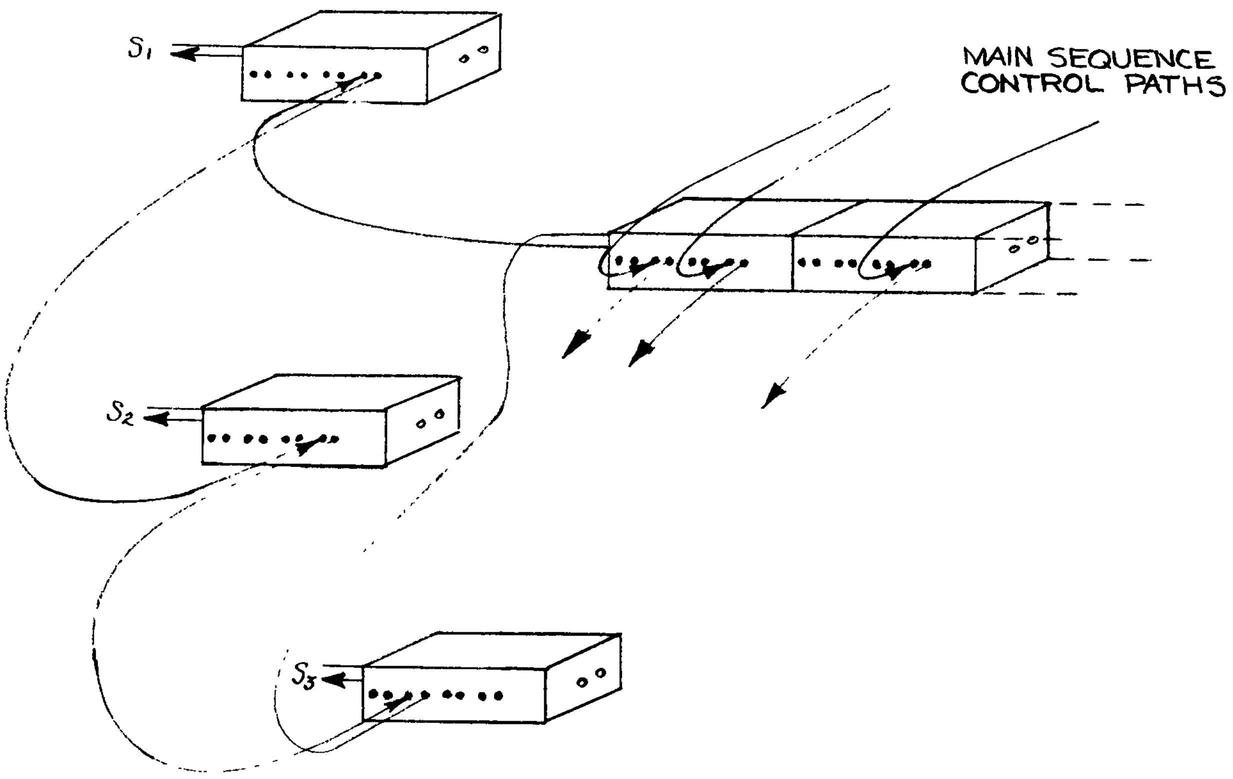


Fig. 39

All of the instructions signal the Call Unit assemblage on the right at the point at which they require an operand. This stack calls on a series of other Call Units, each of which in turn performs a step of the Operand Fetch sub-sequence. After all steps are completed, a signal is returned to the main calling assemblage, from which each instruction's control signal proceeds to initiate succeeding steps defining that particular instruction. Essentially, then, a Call Unit remembers which main control path is calling for the performance of a step or a set of steps during the execution of those steps.

Control Decisions

In order to permit the choice of alternative steps to be made on the basis of data held by the system, two Processing Network elements, a Detector Unit and a Decoder Unit, are provided.

A Detector Unit is used to detect a specific value on a data path. It may be plugged directly onto the unit which provides the data or it may be connected via a Data Cable to a source of data. A data input connector on its base is provided for this purpose.

The binary value to be detected is entered in a set of 12 switches on the unit. A third setting of each switch allows one to indicate indifference to the value of the bit in that position.

A Detector Unit has three control terminals, one for interrogation and the others to indicate the result. When a control signal interrogates the Detector Unit, a data validity check is first performed and the data is then compared with the pattern set in the switches. If the pattern matches the data, a control signal will be presented at the "Yes" terminal. If the pattern does not match the data, a control signal will be presented at the "No" terminal.

For example, suppose that whenever register A contains the binary number 101110 in its rightmost 6-bit positions it is desired to perform some step X, whereas, otherwise, step Y is to be performed. A Detector Unit must be set to detect the desired pattern. The leftmost six switches are set to indicate indifference. The remaining switches are set to read 101110. The Detector Unit is then connected to Register Module A, and the control terminals are connected as shown in Figure 40.

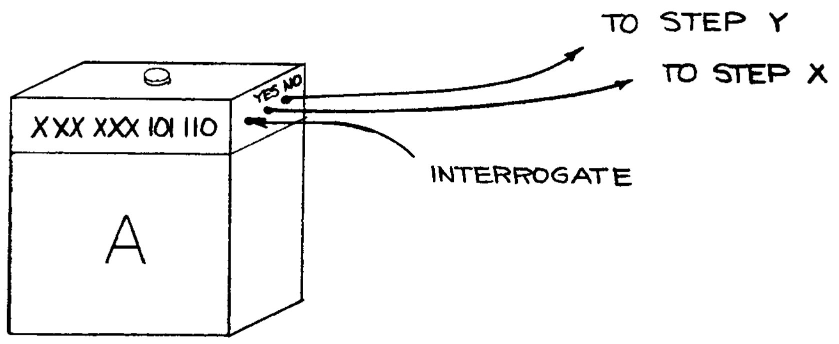


Fig. 40

The Detector Unit, in addition to the above function, passes the data from its input connector through the unit to a data output connector on its top surface, thereby making the register output information available for other possible uses.

Detector Units can thus be stacked, making it convenient to test for any of a variety of possible patterns of interest. Figure 41 shows three Detector Units connected to a single Register Module. They are stacked on top of one another, but in this case they are shown at a distance from the source register and hence are connected to it by a Data Cable. The cable extending above the Detector Unit stack carries the data from A on to other modules.

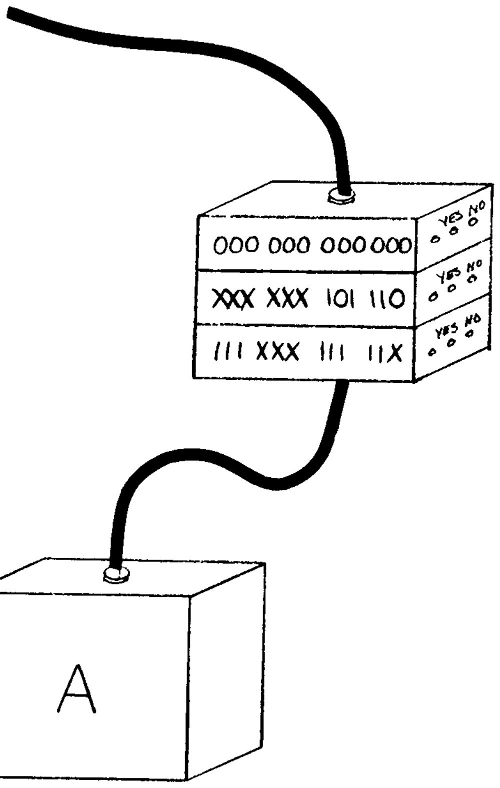


Fig. 41

Detector Units can be plugged together laterally for the detection of patterns of more than 12 bits (Fig. 42). The set of control terminals on the rightmost unit serves for the entire assemblage. No detection decision is made until the data from all sources has been validated.

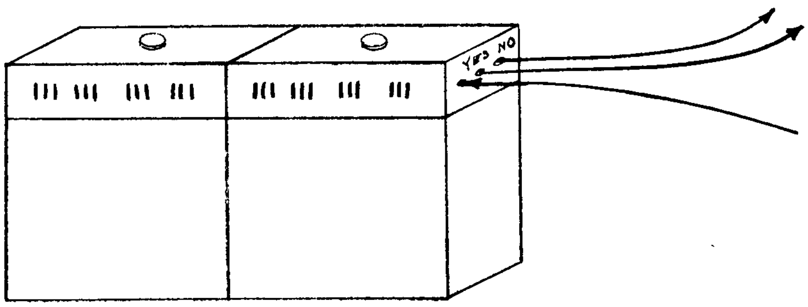


Fig. 42

Figure 43 shows an arrangement which detects a pattern of 24 bits contained in two separate 12-bit registers, A and B.

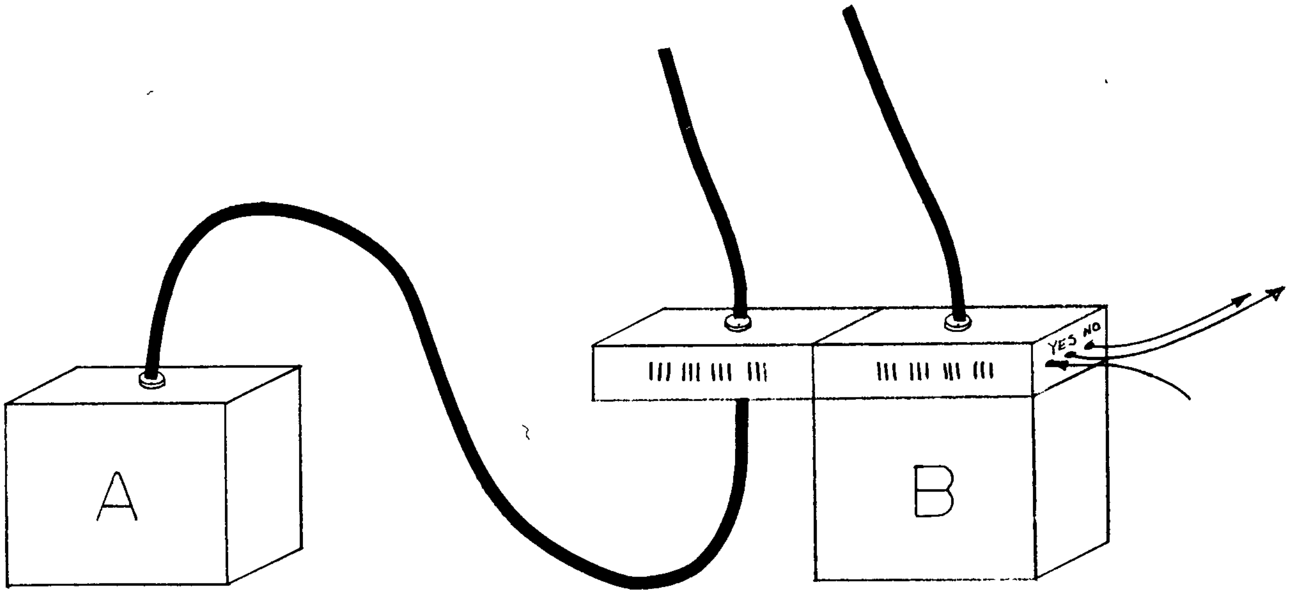


Fig. 43

Figure 44 shows an arrangement which detects a pattern of 24 bits from a variety of sources.

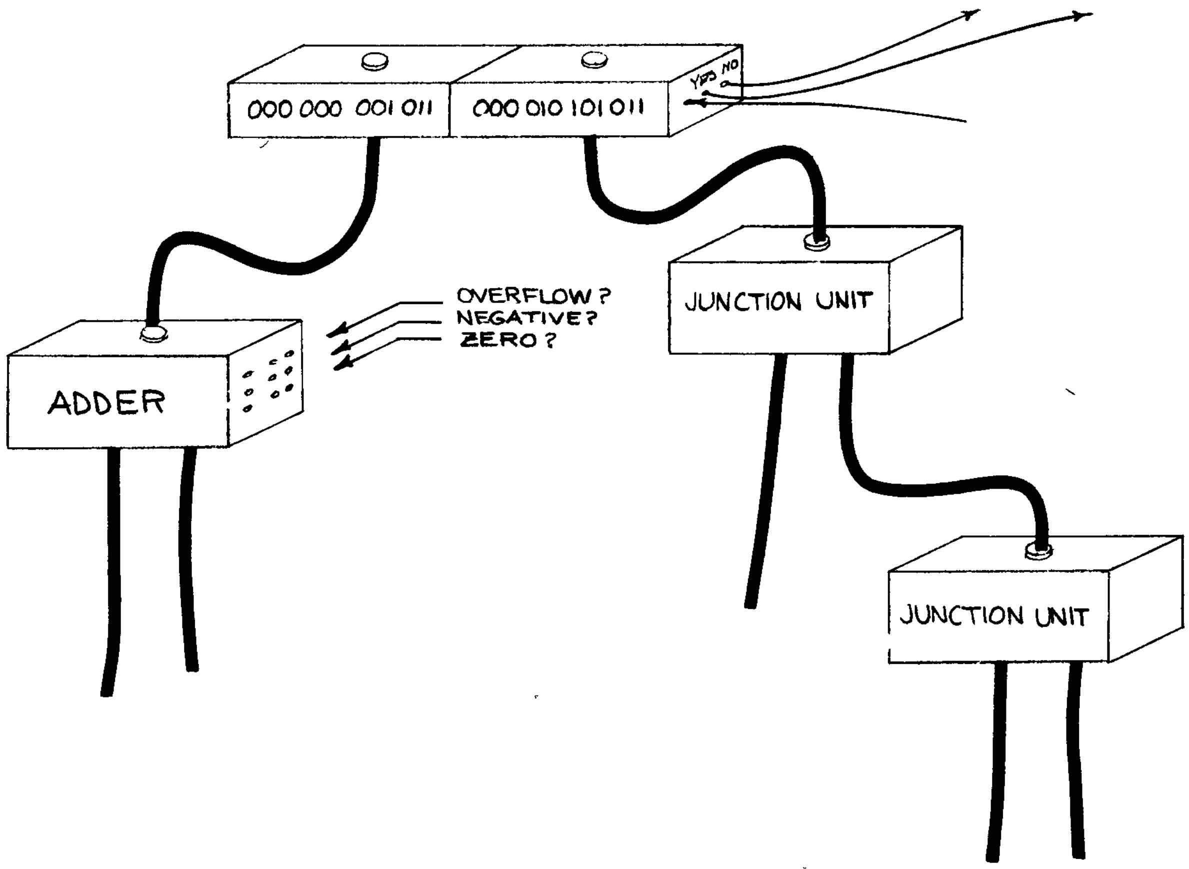


Fig. 44

The Adder Unit, as mentioned earlier, has three sets of detector terminals. These terminals are similar to the control terminals on the Detector Unit and are used in an equivalent manner for detection of carry overflow, negativity, and the numerical sum zero. A signal returned at a "Yes" terminal indicates that the associated condition exists.

Detector Units make it possible to select one of two alternative control paths on the basis of particular data values or patterns. Sometimes, however, it is desirable to select one of 2^n paths on the basis of n bits of data. In this case 2^n detectors might be used, but the number of detectors required grows rapidly with n. Instead, therefore, a Decoder Unit is provided (Fig. 45). This unit contains a 3-bit decoder which may be interrogated by a control signal.

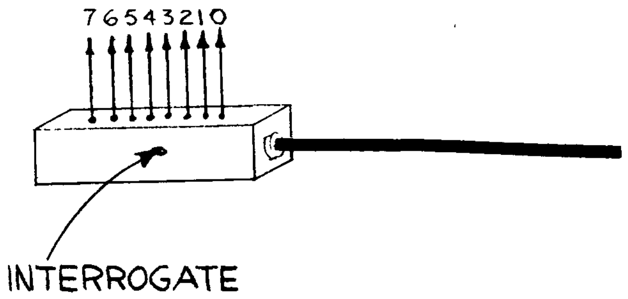


Fig. 45

The cable shown entering the side of the unit is a standard Data Cable coming from some source. The input data is passed through the detector to a data output connector on the opposite surface of the unit. On the face of the Decoder Unit, switches are provided for selecting three of the 12 data lines for decoding. When a control signal arrives at the interrogate terminal (also on the front face of the unit), the data is validated and then a signal is produced on one of the eight output terminals (shown on top). Each output terminal corresponds to one of the eight possible values encoded on the selected bits, and the signal appears at the corresponding terminal.

To permit decoding of values encoded on fewer bits, the switches for bit selection include a position which supplies an apparent "0" to the decoder. If, for example, the most significant bit of the decoded subset is thus fixed, an output signal will never appear on lines 4, 5, 6, or 7.

To illustrate the technique of decoding values encoded on a larger number of bits, let us consider the problem of selecting one of 32 alternative control paths. To be explicit, let us label the 12 data lines X_0, X_1, . . . X_{11} and let us suppose that we wish to decode the value on lines X_0 - X_4. Figure 46 shows the necessary connections.

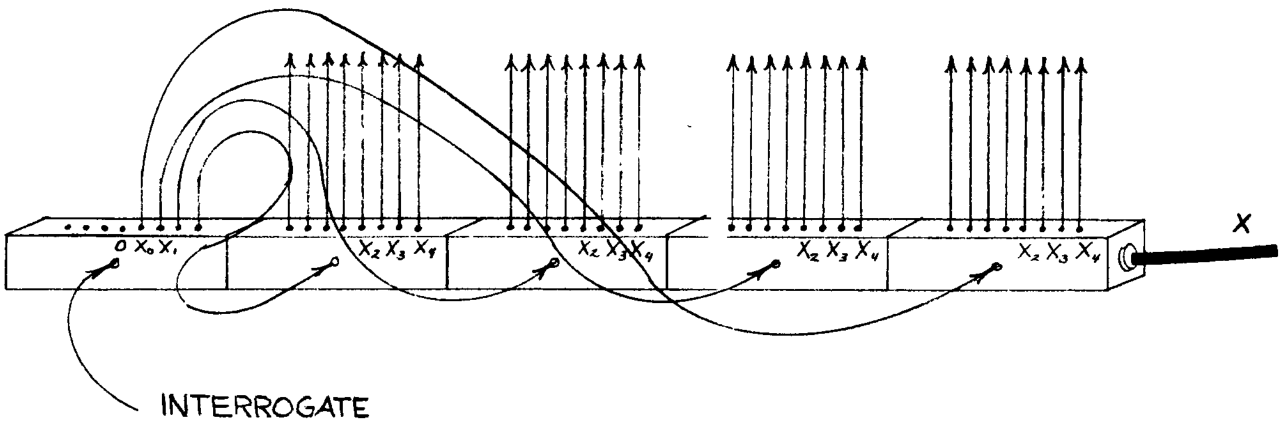


Fig. 46

Note, first of all, that the Detector Units can be plugged together laterally. This passes the data on the X data cable to all of the units. Each unit is labeled to indicate which lines are selected for decoding. The leftmost unit decodes the value on two lines, X_0 and X_1. When the interrogate signal arrives at the leftmost unit, a signal will appear on one of four output lines as shown, depending on the states of X_0 and X_1. This signal is routed to the interrogate terminal of one of the other Decoder Units which, based on the values on X_2, X_3, and X_4, in turn produces a signal on one of its eight output lines. Extensions of this technique make it possible to decode data values encoded on any number of bits.

Merge Unit

At some point after making a decision, all of the decision-dependent steps will have been executed, and the corresponding alternate control paths may be joined through the use of a Merge Unit.* This unit produces a signal at its output terminal whenever a signal appears at either input terminal (Fig. 47).

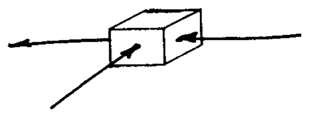


Fig. 47

Figure 48 shows these units connected to merge four control paths. An input signal on any of lines 1, 2, 3, or 4 will produce an output signal on line 5. The units may also be plugged directly together laterally as shown in Fig. 49.

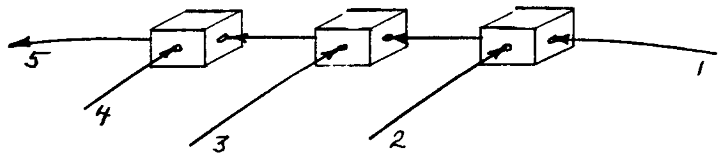


Fig. 48

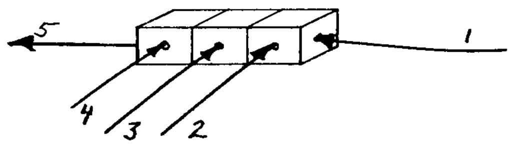


Fig. 49

* This unit, like the Control Branch, may be used to interconnect and thereby extend Control Cables.

Decision Call Unit

A Decision Call Unit permits a detector unit to be accessed by more than one control path and contains five decision calling elements as shown in Fig. 50.

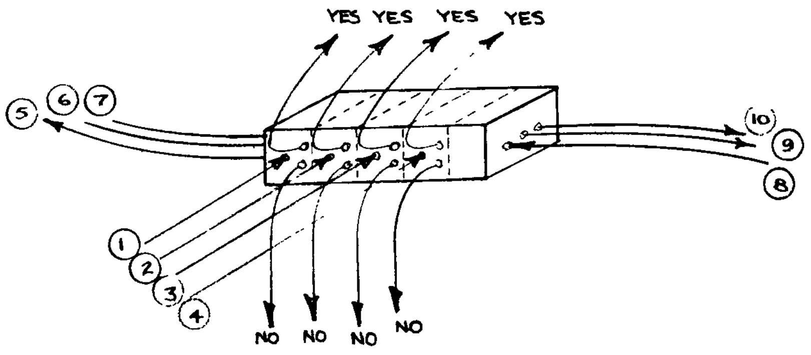


Fig. 50

This unit plugs directly onto the control terminals of a Detector Unit, terminal 5 mating with the interrogate terminal and terminals 6 and 7 with the "Yes" and "No" terminals as shown below.

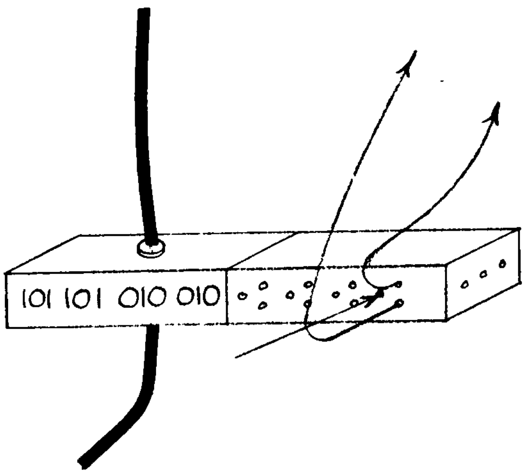


Fig. 51

When a control signal is presented at the input of any of the five decision calling elements, a signal is produced at terminal 5 which interrogates the Detector Unit. A "Yes" or "No" signal is returned to the Decision Call Unit and will appear at the "Yes" or "No" terminal of the element which called for the interrogation.

Decision Call Units can be plugged together to allow an arbitrarily large number of control paths to access the same Detector Unit. Decision Call Units may also be connected to the Detector Unit by means of control cables as shown in Fig. 52.

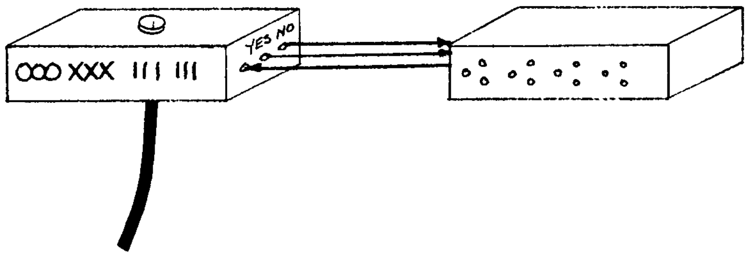


Fig. 52

Like the Call Unit, the Decision Call Unit may be used to provide multiple access to a sub-sequence control path. In this case, the subsequence may include a decision in which one of two alternative control paths is selected. In this context the Decision Call Unit may be thought of as calling a subroutine which has two return points. A simple example of this is shown in Fig. 53. Register C is used as a counter which, whenever indexed, is checked for the value 7777_8. The use of the Decision Call Unit allows this counter to be used by many different control paths. Obviously any number of steps may occur in the sub-sequence prior to or following the decision.

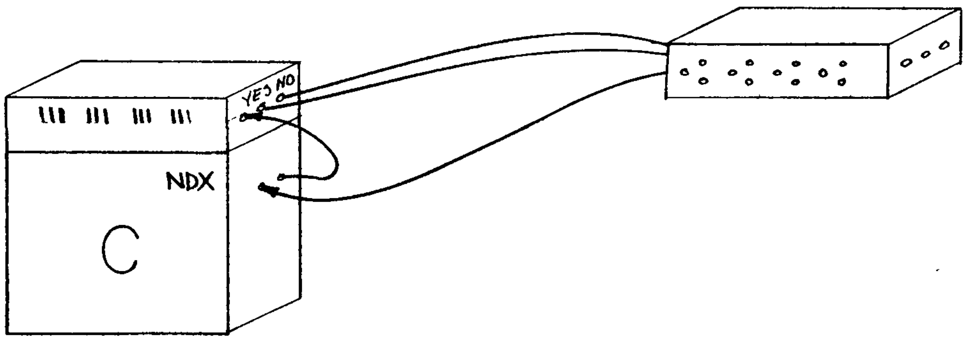


Fig. 53

Interlocking

In some situations two independent sequences will both require the use of the same data-processing element(s), (e.g., two sequences which make use of the same memory), and conflicts may arise. For such situations an Interlock Unit (Fig. 54) is provided. This unit sorts incoming control signals on a "first-come, first-served" basis, interlocking them in such a way as to resolve conflicts.

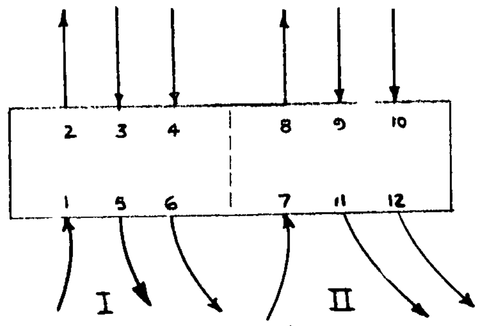


Fig. 54

The left and right halves are associated with the control paths, I and II, of two concurrent sequences which must be interlocked. For a sequence to enter an interlocked phase, a signal must be presented to the interlock at an input terminal (terminal 1 for I, terminal 7 for II). Because it must be assumed that the sequences do not necessarily contain the same steps within their interlocked phases, each control path is provided with its own terminals (terminals 2, 3, 4 for I; terminals 8, 9, 10 for II) for use during the interlocked phase. If the interlock is off when a signal arrives at terminal 1, it is turned on and a signal is produced at terminal 2. This signal is used to initiate the steps within the interlocked phase of the sequence associated with control path I. After the last of these steps has been completed, a signal is returned to terminal 3 or terminal 4. (Two return terminals are provided to allow for a possible decision within the interlocked section. If no such decision is required, either terminal 3 or 4 may be used for the return.) In either case, the return produces a signal at either terminal 5 or 6, depending on whether the return came to terminal 3 or 4, and shuts off the interlock. An equivalent process takes place for control path II, using terminals 7 through 12. If either control signal enters the interlock while it is on, it will be held up until the interlock is turned off. If signals arrive at terminals 1 and 7 simultaneously, only one will be accommodated immediately; the other will wait its turn.

Figure 55 shows an arrangement for interlocking two sequences, both of which use register A.

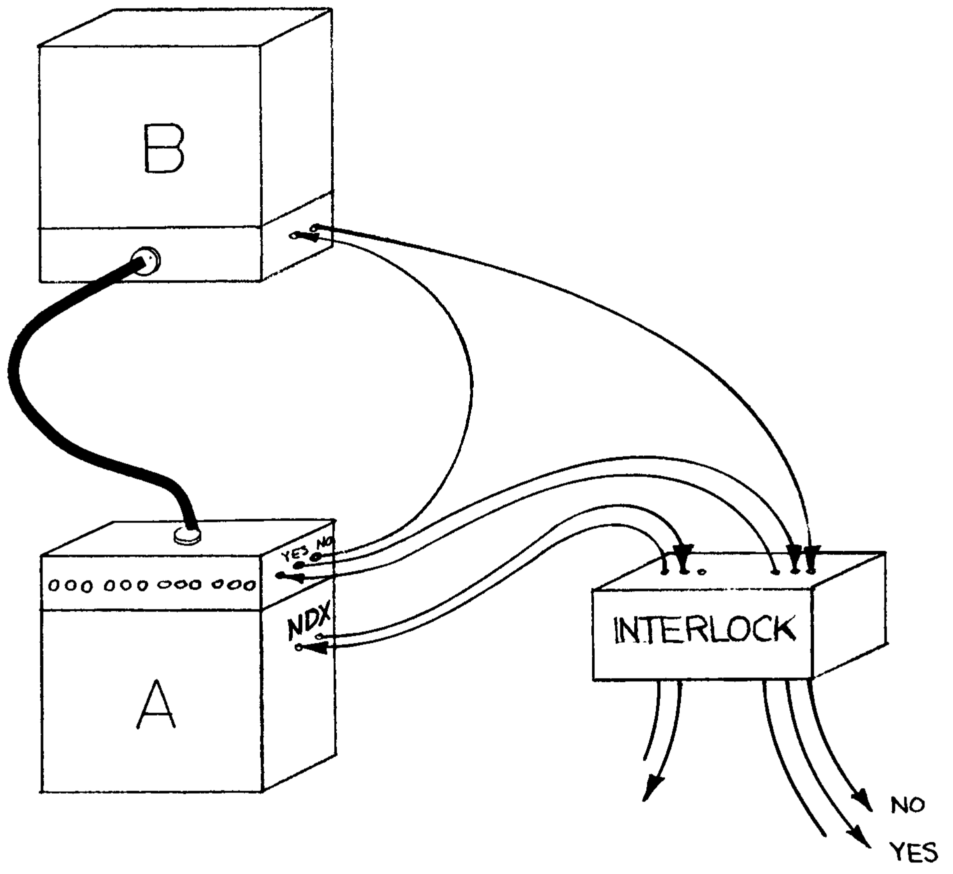


Fig. 55

Interlock Units can be plugged together (Fig. 56) to permit interlocking of any number of sequences.

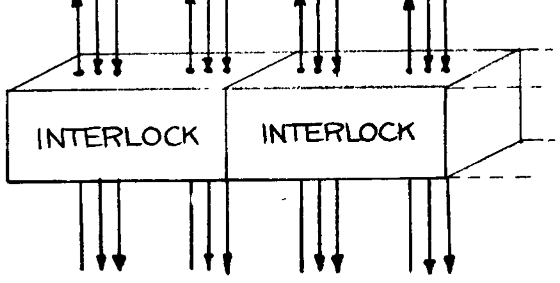


Fig. 56

EXAMPLES

Two examples of macromodular structures are now presented. The first is a simple multiplier. The second is a "central processor" for a small computer.

Example No. 1 - A Multiplier

In this example, we will consider the multiplication of numbers with a positive sign and eleven bits of magnitude. The multiplier and the multiplicand are held in registers B and X, respectively, and the product is to appear in register A. Figure 57 shows the processing structure required.

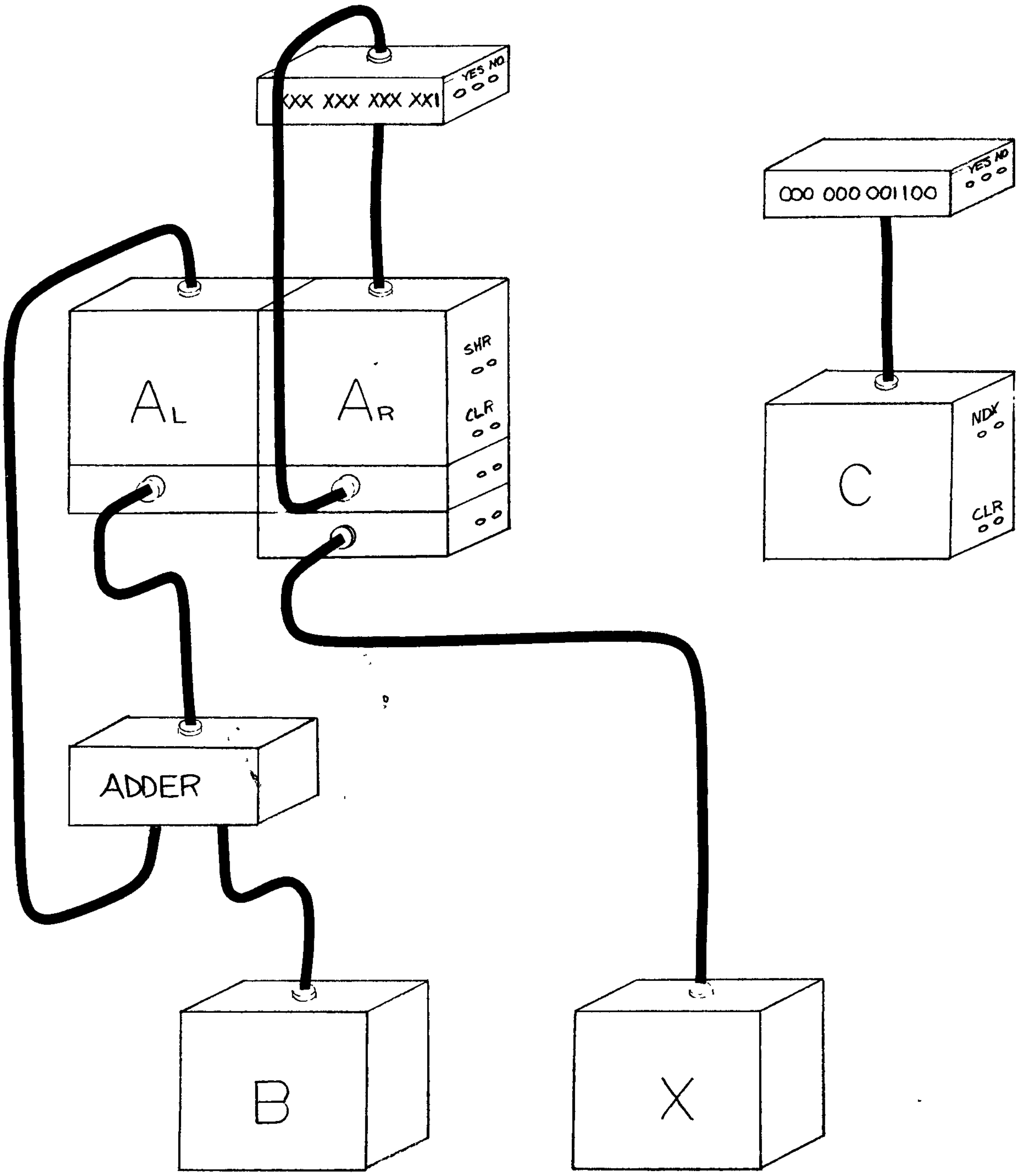
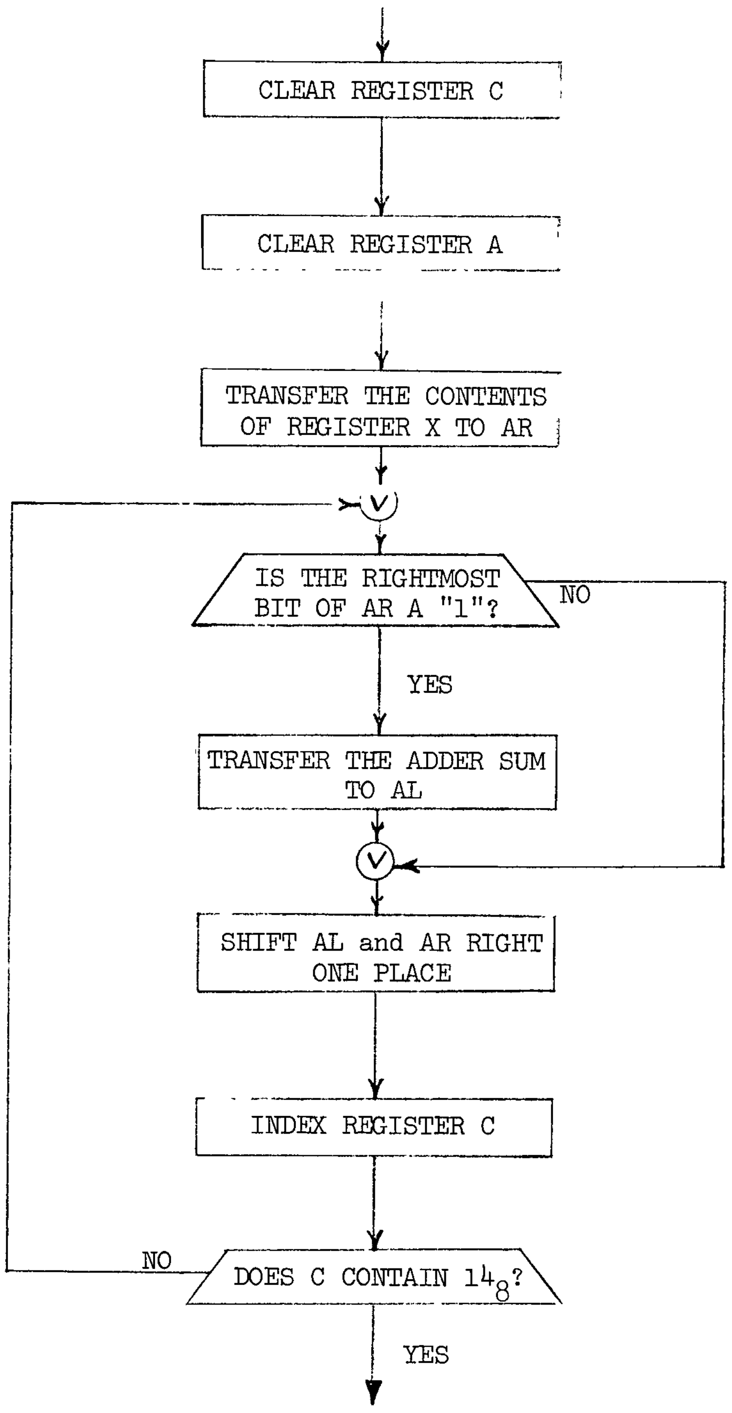


Fig. 57

Figure 58 shows the algorithm to be used[5]. This algorithm calls for a succession of shifts and conditional additions while register C counts the number of times these steps are repeated. The process terminates when twelve (14_8) cycles have been completed.



CLEAR REGISTER C
CLEAR REGISTER A
TRANSFER THE CONTENTS OF REGISTER X TO AR
IS THE RIGHTMOST BIT OF AR A "1"?
NO
YES
TRANSFER THE ADDER SUM TO AL
SHIFT AL and AR RIGHT ONE PLACE
INDEX REGISTER C
NO
DOES C CONTAIN 14_8?
YES

Fig. 58

Figure 59 shows the addition of the sequencing network which controls the processing structure in such a way as to perform the multiplication according to the algorithm. The multiplication is performed whenever a signal is presented on line 1. A signal on line 2 marks the completion of the multiplication. Careful inspection reveals that the control lines shown have a one-to-one correspondence with the arrows which appear on the flow diagram. At the terminus of each control line, one of the functions described in the flow diagram boxes is performed.

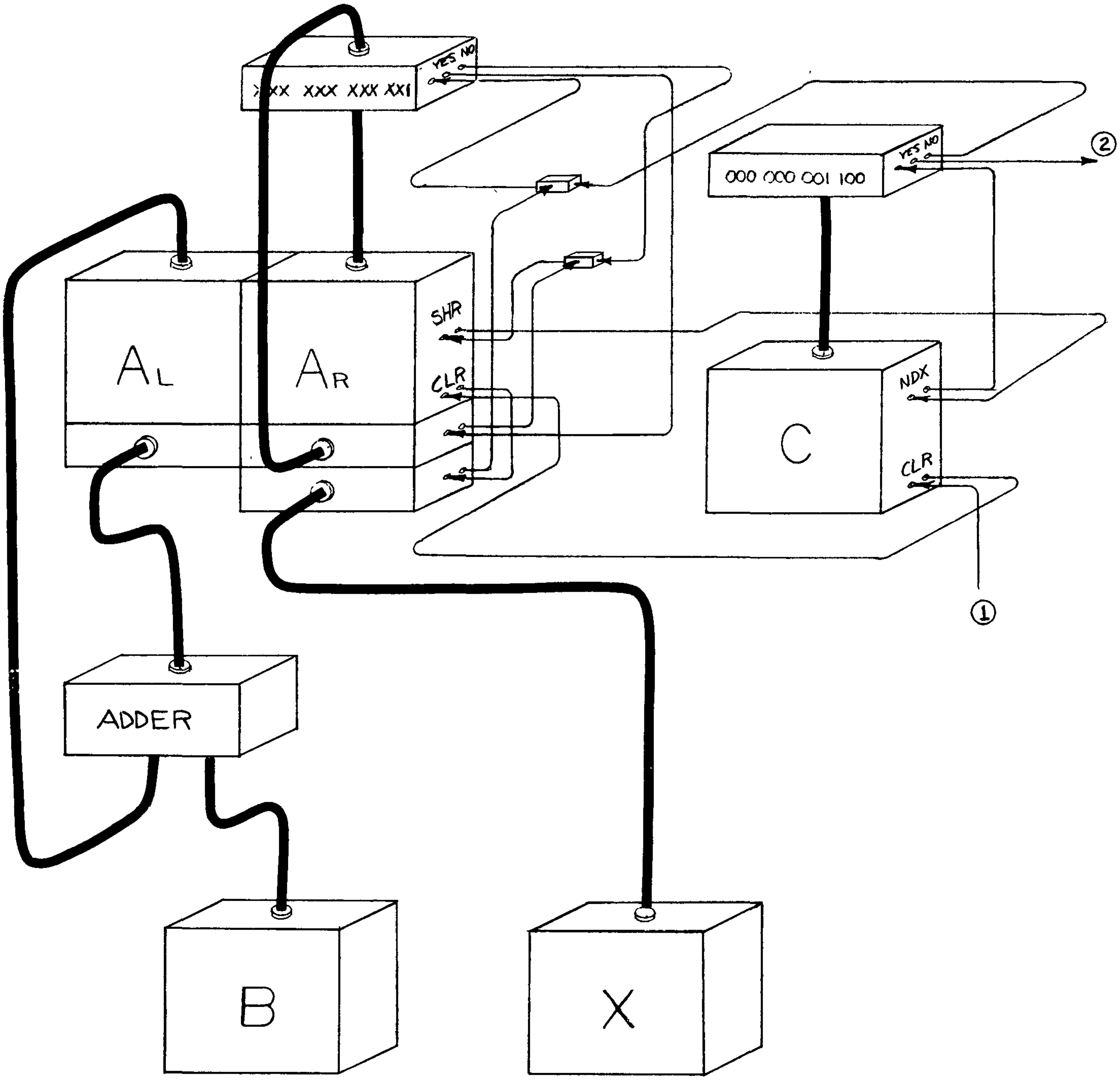


Fig. 59

Example No. 2 - A Small Computer

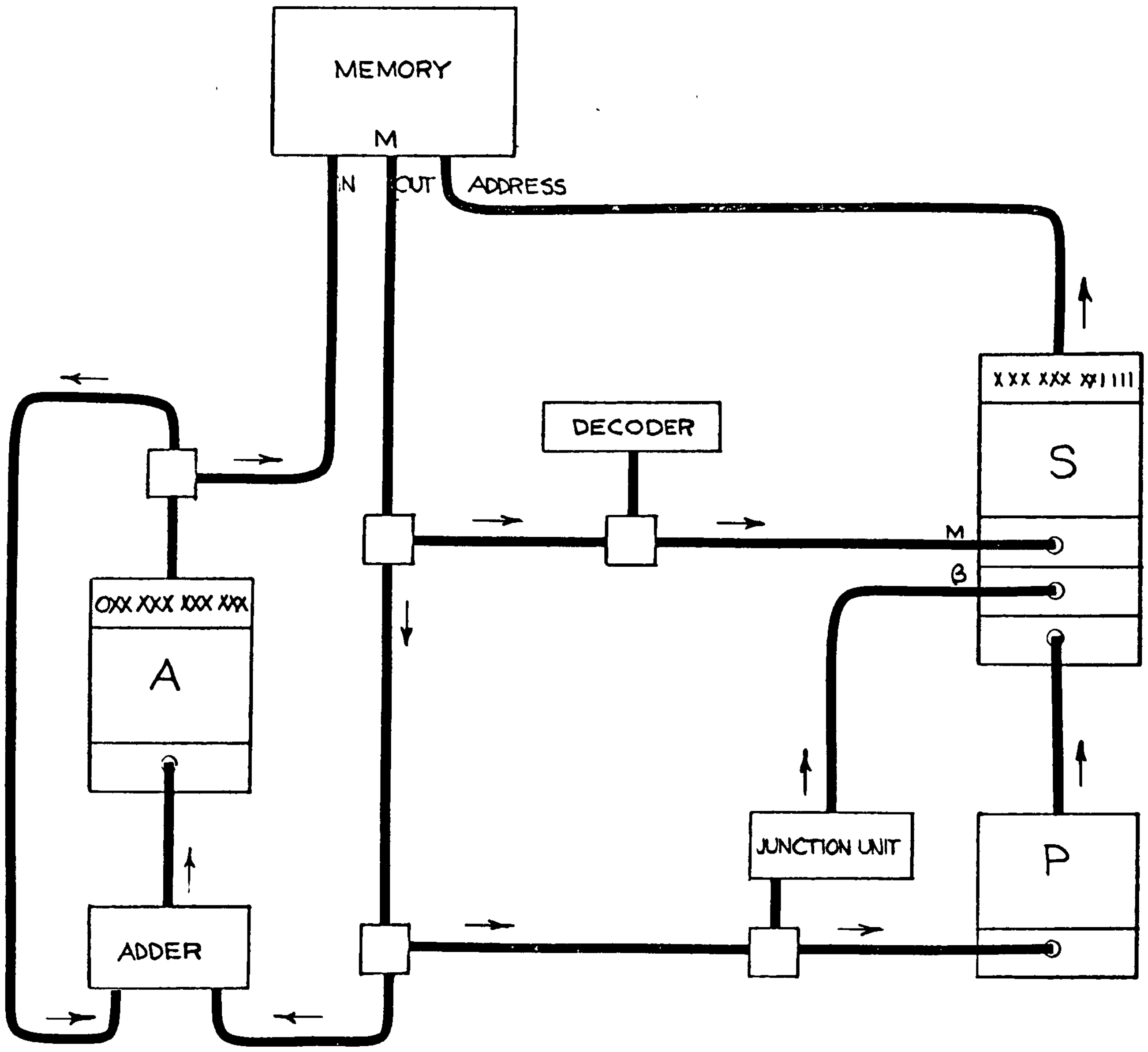
Let us consider the central processing portion of a very simple computer and sketch out how it might be realized in macromodular form. The computer will have a 12-bit word length, 4096 words of programmable memory, and an instruction repertoire consisting of eight instructions encoded on the leftmost three bits of the 12-bit instruction word.

Three of the instructions make reference to other memory locations by a process of indirect addressing. These instructions are ADD β, STO β, and JMP β and use the contents of memory register β $(0 \leq \beta \leq 777_8)$ as the effective address. Thus, for example, if register β contains the number 1476, execution of an ADD β instruction will cause the contents of register 1476 to be added to the contents of the accumulator (in ones' complement form). Similarly, a STO β will store the contents of the accumulator in register 1476. A JMP β will cause the next instruction to be taken from location 1476.

The remaining five instructions do not make reference to other memory locations. Instead, they perform the following functions:

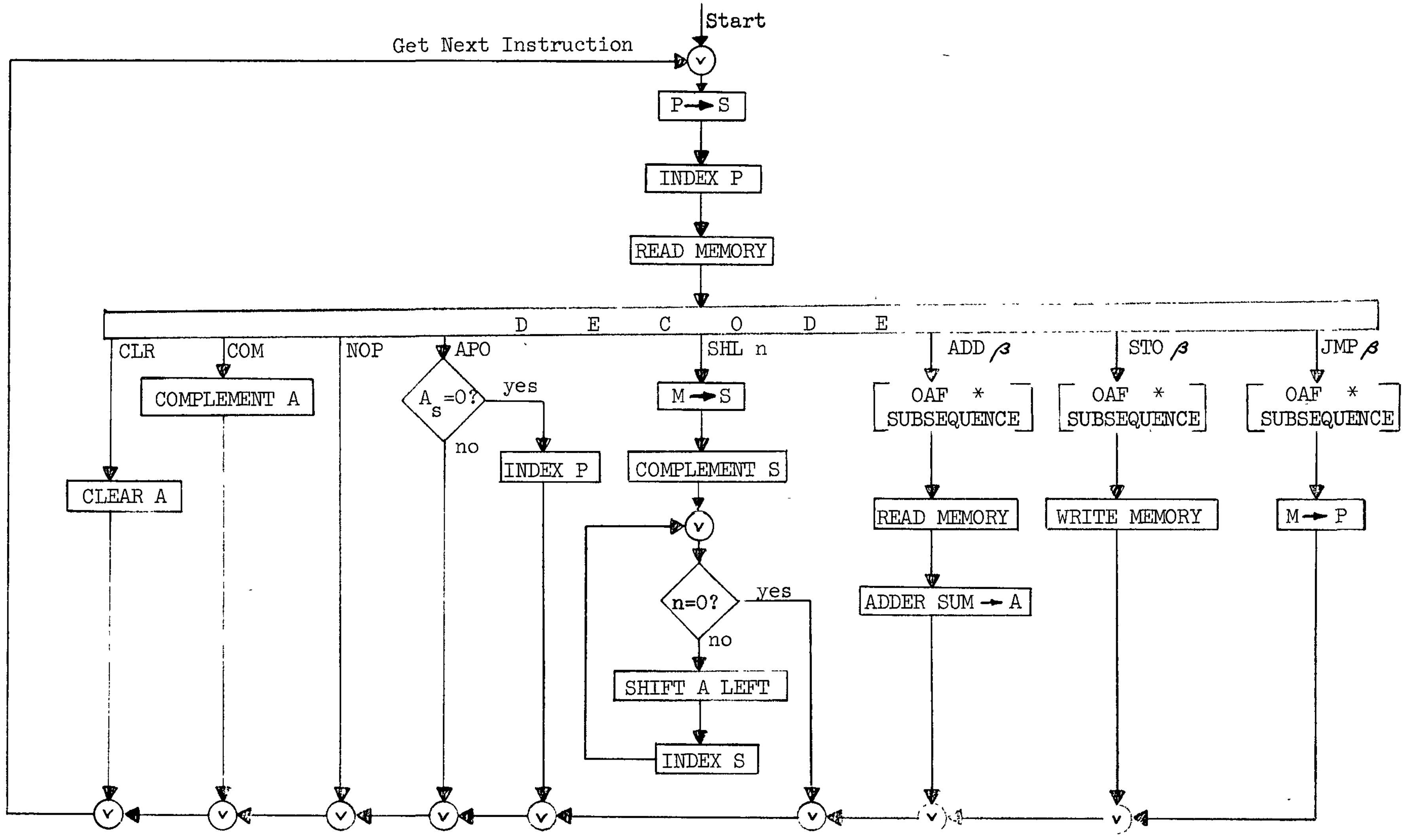
CLR	-	Clear the accumulator.
COM	-	Complement the contents of the accumulator.
APO	-	Skip the next instruction if the accumulator contains a positive number.
SHL n	-	Shift the contents of the accumulator n places to the left where n is specified by the four rightmost bits of the instruction word.
NOP	-	Proceed immediately to the next instruction.

Figure 60 gives a two dimensional view of the processing network for this machine. Register A is the accumulator; register S is used to provide addresses to the memory and also to count out the number of shifts required by an SHL n instruction. During the execution of an instruction, register P holds the address of the next instruction to be executed. The Decoder Unit is set to decode the instruction field (3 most significant bits) of the word from memory, (M). The Junction Unit is set to mask out the instruction field when transferring β to S.



MEMORY
M
IN
OUT
ADDRESS
XXX XXX XX1111
S
M
B
DECODER
OXX XXX XXX XXX
A
ADDER
JUNCTION UNIT
P

Fig. 60


Start
Get Next Instruction
P → S
INDEX P
READ MEMORY
D E C O D E
CLR
COM
NOP
APO
SHL n
ADD β
STO β
JMP β
COMPLEMENT A
A_s=0?
yes
no
INDEX P
CLEAR A
M → S
COMPLEMENT S
n=0?
yes
no
SHIFT A LEFT
INDEX S
OAF * SUBSEQUENCE
OAF * SUBSEQUENCE
OAF * SUBSEQUENCE
READ MEMORY
ADDER SUM → A
WRITE MEMORY
M → P

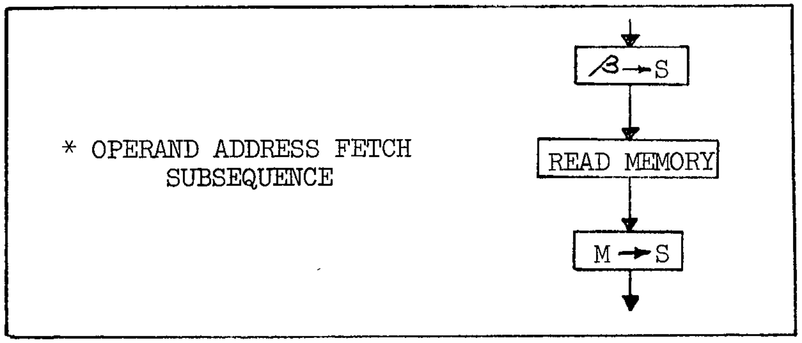

* OPERAND ADDRESS FETCH SUBSEQUENCE
β → S
READ MEMORY
M → S

Fig. 61

Figure 61 shows a flow diagram representation of the operation of the machine. Entering the top of the flow diagram corresponds to getting the next instruction from the memory. The contents of P are transferred to S in order to locate the instruction, and P is then indexed in preparation for locating the following instruction. Memory is read to obtain the instruction and the decoder is signaled to decode the instruction field of the memory word. Eight decoder output control paths are provided, one for each of the instructions, and a control signal is produced on the selected path. After the required sequence of steps has been executed, the control signal returns to get the next instruction, (GNI).

Figure 62 shows the control paths required to locate and decode the instructions, as well as the execution control paths for the instructions CLR and COM. Figure 63 shows the execution control paths for the instructions APO and NOP. The detector in the APO control path is set to detect a "0" in the leftmost (sign) position of the accumulator. A "No" signal response from the detector proceeds to the Merge Unit assemblage from which a signal to get the next instruction emerges. A "Yes" response indexes register P and then proceeds to get the next instruction. The NOP instruction has no execution steps, and its control path is routed directly to the GNI merge unit assembly.

The SHL n instruction, shown in Fig. 64, uses the S register to count the number of places which have been shifted. The SHL n instruction word is transferred from the memory into S and complemented. A detector on S monitors the rightmost four bits, (n), and provides a "Yes" response when the value 17_8 (the minus zero in ones' complement form) is detected. A control loop is entered in which the detector is interrogated. A "No" signal response from the detector shifts A one place, indexes the count in S, and again checks the detector. This process repeats as long as more shifts are required. When the required number of shifts has been completed, interrogation of the detector will produce a signal at its "Yes" output, which is routed to the GNI Merge Unit assemblage.

The control paths for the three memory reference instructions are routed to a sub-sequence Call Unit. This sub-sequence fetches the operand address and places it in the S register. This three-step sub-sequence is shown separately on the flow diagram in Fig. 61, and the control path which realizes it is shown on Fig. 65. The first step of the sub-sequence is the transfer of β to S. In the following steps, the effective address is obtained from

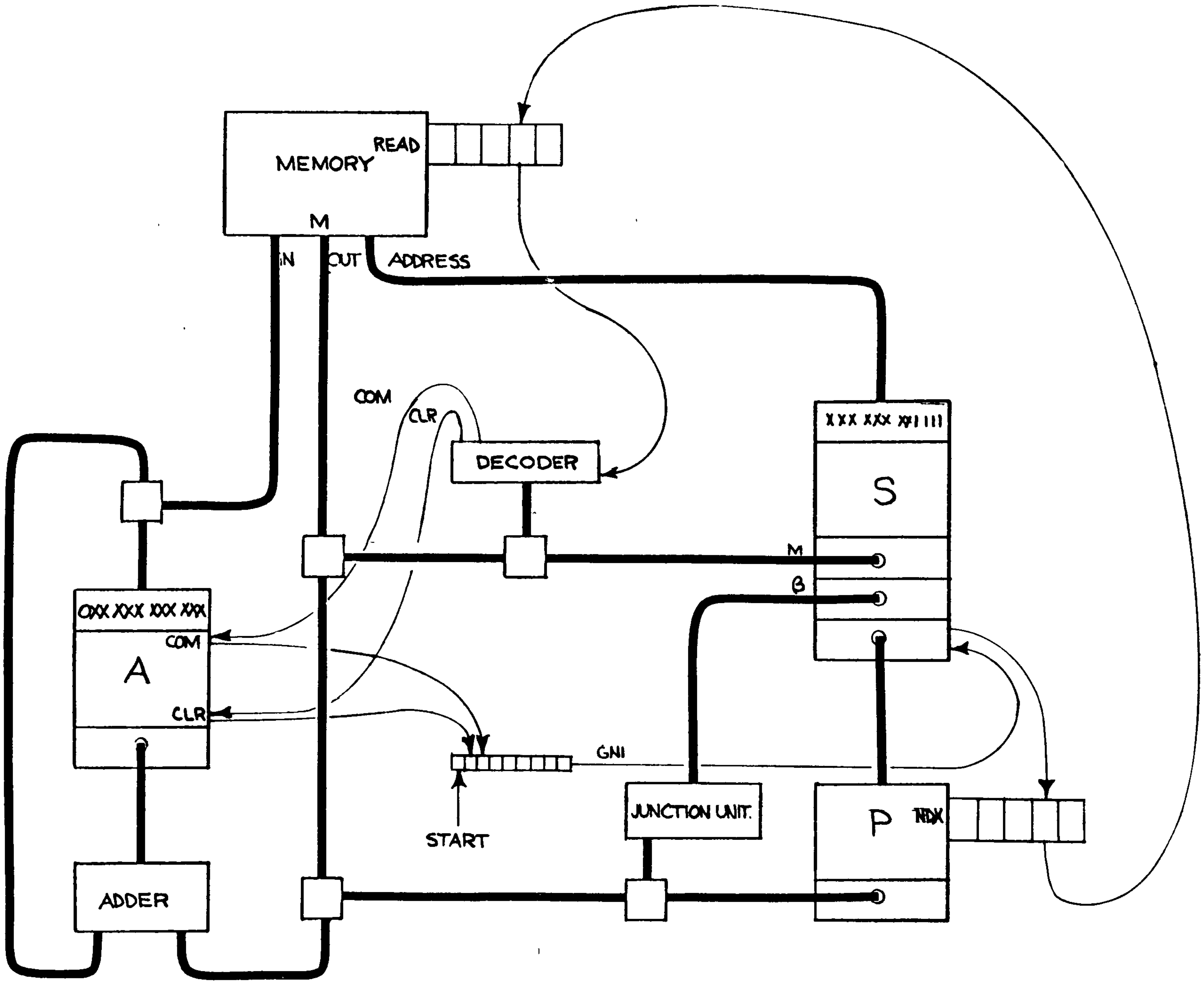
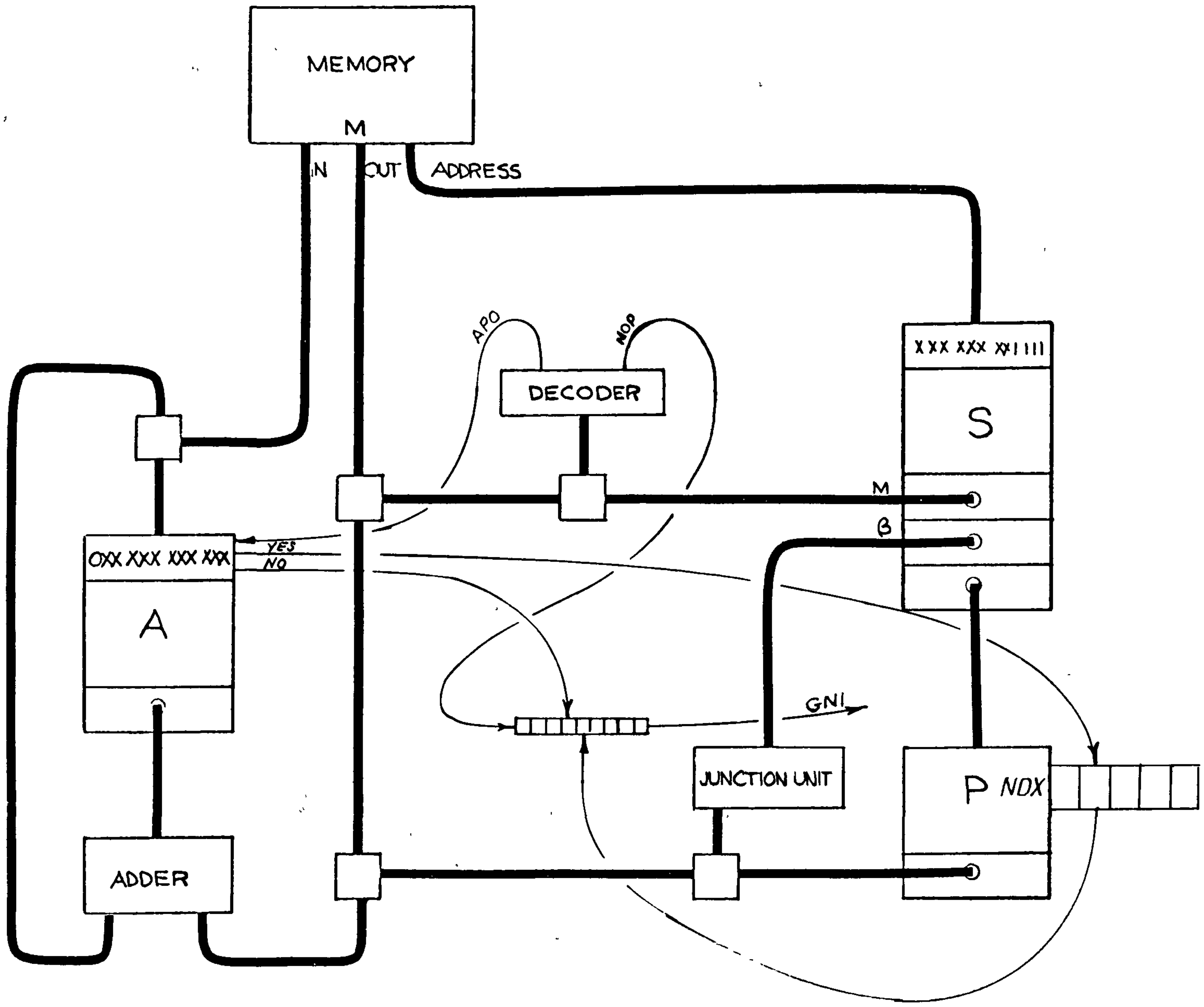

MEMORY
READ
M
IN
OUT
ADDRESS
COM
CLR
DECODER
XXX XXX XX1111
S
M
B
0XX XXX XXX XXX
COM
A
CLR
GNI
START
JUNCTION UNIT.
P
NDX
ADDER

Fig. 62



MEMORY
M
IN
OUT
ADDRESS
APO
NOP
DECODER
XXX XXX XX1111
S
M
B
OXX XXX XXX XXX
YES
NO
A
GNI
JUNCTION UNIT
P NDX
ADDER

Fig. 63

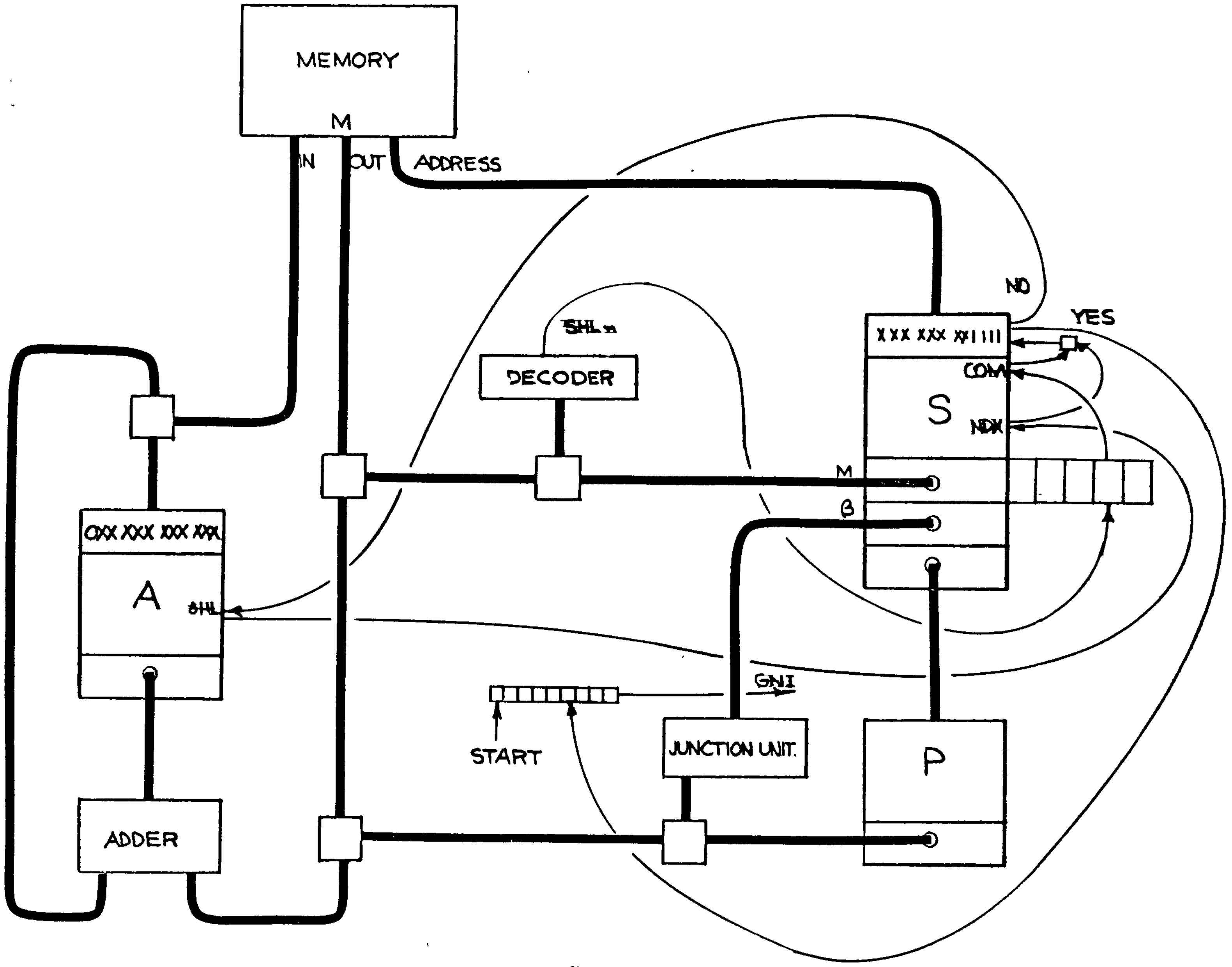

MEMORY
M
IN
OUT
ADDRESS
NO
YES
XXX XXX XX1111
COM
S
NDX
M
B
SHL n
DECODER
0XX XXX XXX XXX
A
SHL
ADDER
GNI
START
JUNCTION UNIT.
P

Fig. 64

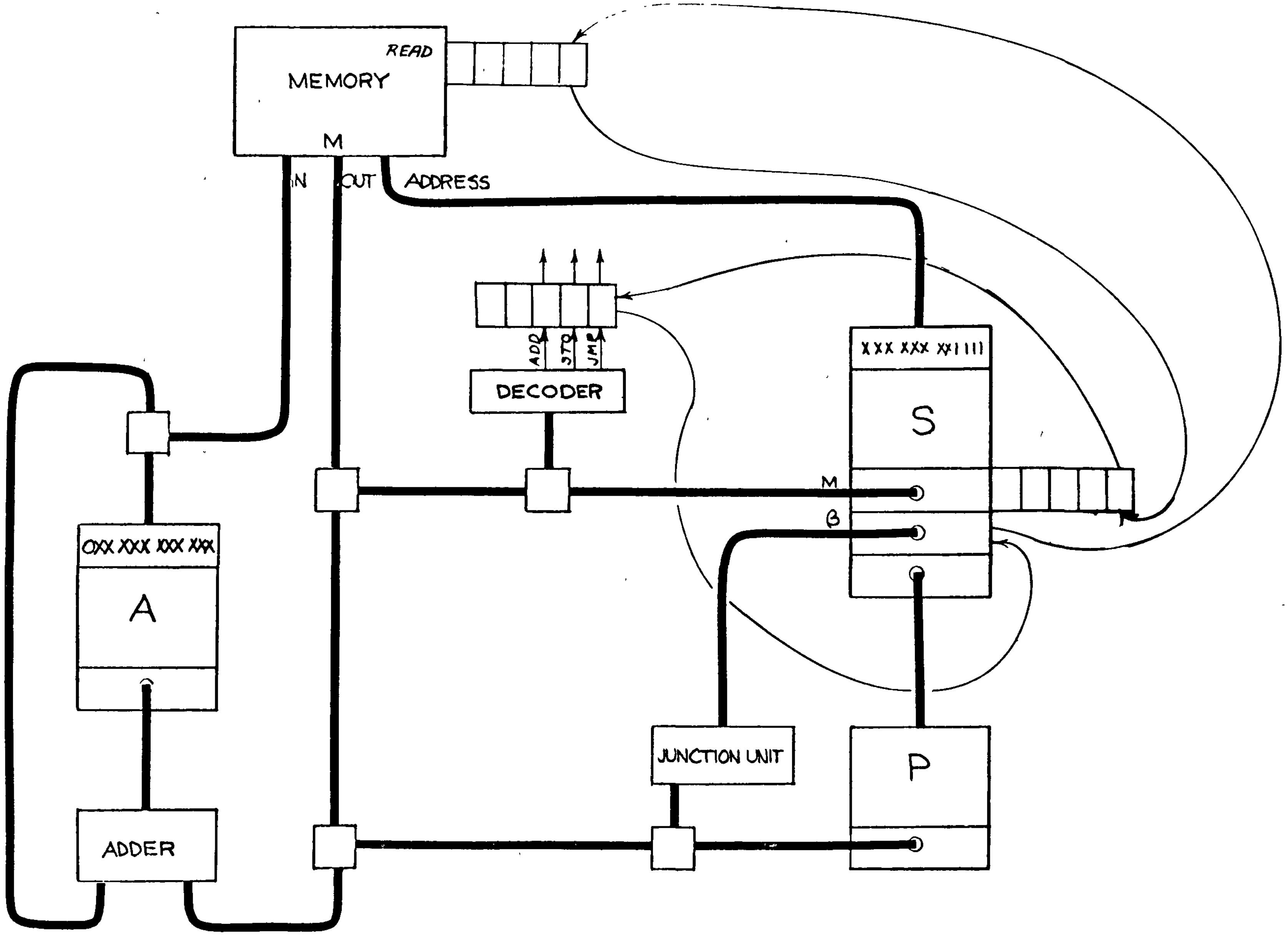

READ
MEMORY
M
IN
OUT
ADDRESS
ADD
STO
JMP
DECODER
XXX XXX XX1111
S
M
B
0XX XXX XXX XXX
A
ADDER
JUNCTION UNIT
P

Fig. 65

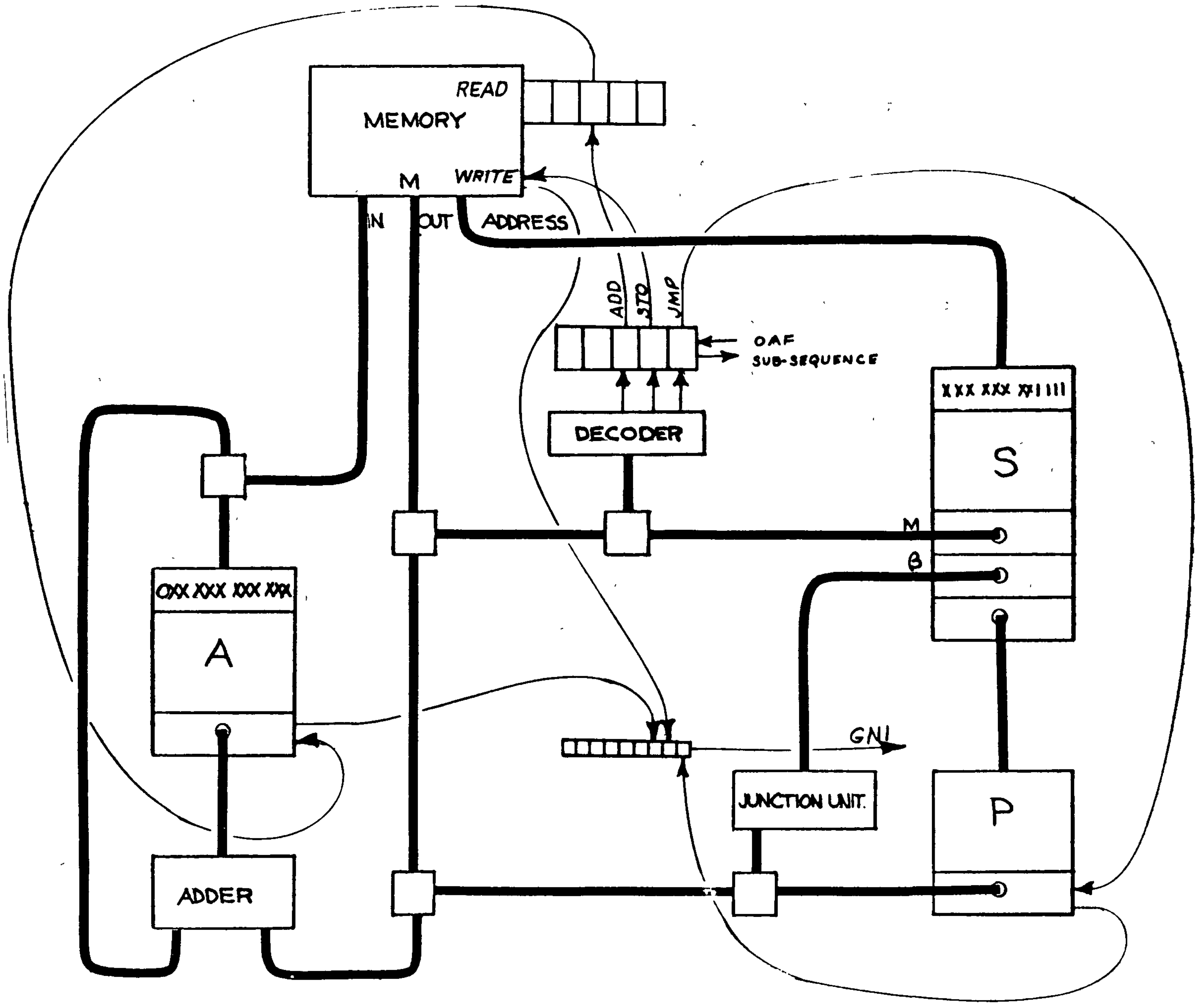

READ
MEMORY
M
WRITE
IN
OUT
ADDRESS
ADD
STO
JMP
OAF
SUB-SEQUENCE
DECODER
XXX XXX X11111
S
M
B
0XX XXX XXX XXX
A
GNI
JUNCTION UNIT.
P
ADDER

Fig. 66

memory register β and transferred into S. Beyond this point the three instructions' control paths diverge, as shown on Fig. 66.

In the ADD instruction the memory is again read to obtain the operand. The adder forms the sum of this operand and the number in the accumulator, and in the next step of the sequence, the sum is transferred into A.

The STO instruction execution sequence consists simply of a single Write command to the memory, as Register A is connected directly to the memory Data Input.

In the JMP instruction, the effective address is transferred into P from the memory. For this instruction the final step of the operand address fetch subsequence (which put the operand address into S) was unnecessary but harmless.

This example demonstrates the ease and directness with which simple systems may be put together. A total of 16 Data Cables, 35 Control Cables, and 17 modules or module assemblages are required to form this complete, albeit comparatively modest, central processor.

CONCLUSION

We have presented an approach in which flexibility and simplicity are brought to the forefront of factors relevant in computer design. Two aspects of this approach can be distinguished.

First, it adds an experimental element to the theoretical and simulational techniques now available to the system designer. Perhaps this can most effectively be employed in research groups having as their main objectives the creation of advanced computer systems for particular areas of work. Because of the relative ease with which a given configuration of macromodules can be profoundly altered, it is possible for such a group to work actively with several widely different forms in an attempt to find optimal configurations for different problem classes.

Second, it makes possible a smoothness of growth and refinement in an operating computing system. Because of the electronic independence of the macromodules, it is relatively easy to expand a macromodular system and to add new functions without seriously affecting the continuity of on-going work and without jeopardizing any existing investment in programs and operating procedures.

These properties of macromodular systems are of growing importance as we turn increasingly toward the user for new concepts in the search for more effective information processing systems.

ACKNOWLEDGMENT

The authors wish to express their gratitude to J. R. Cox, Y. H. Chuang, T. J. Chaney, C. E. Molnar, and R. A. Ellis for their help in the development of the ideas presented here. Special thanks are due N. T. Kinch, R. E. Olson, and H. C. Lewis for the preparation of the very difficult figures, and to Mrs. B. Douglas for her conscientious job in preparing the manuscript.

REFERENCES

1. Estrin, G., "Organization of computer systems: the fixed plus variable structure computer," Proc. WJCC, 1960, 33-40.

2. Menabrea, L. F., "Statement of the Circumstances Attending the Invention and Construction of Mr. Babbage's Calculating Engines," Philosophical Magazine, Sept. 1848, p. 235.

3. Andrews, E. G., "The Bell Computer, Model VI," Annals of the Computation Laboratory of Harvard University, Harvard University Press, 1951, Vol. VI, pp 20-31.

4. Muller, D. E., "Asynchronous Switching Theory,"-Digital Computer Laboratory Report 243, University of Illinois, June, 1958.

5. Mann, M. F., Rathbone, R. R., and Bennett, J. B., "Whirlwind I Operation Logic," Digital Computer Laboratory Report R-221, Massachusetts Institute of Technology, May 1954, pp 3.16 - 3.18.

MACROMODULAR COMPUTER SYSTEMS

Technical Report No. 4

June 20, 1967

Computer Systems Laboratory
Washington University
St. Louis, Mo.

Reprints of a series of papers presented at the 1967 Spring Joint Computer Conference sponsored by the American Federation of Information Processing Societies at Atlantic City, April 19, 1967, as published in the Proceedings, Thompson Book Co., National Press Building, Washington, D.C. 2004. (Only the first three articles are reprinted here.)

The research reported in these papers was supported in part by the Information Processing Techniques Project of the Advanced Research Projects Agency under Contract SD-302, and by the Division of Research Facilities and Resources of the National Institutes of Health under grant FR-00218.

TABLE OF CONTENTS

Macromodular computer systems*

by WESLEY A. CLARK
Chairman's Introduction
Washington University
St. Louis, Missouri

INTRODUCTION

The amount of logically irrelevent engineering detail inherent in the design and construction of a computer system is great. As a result, the task of creating a system based on the use of present techniques is so difficult and time-consuming that the number of different systems that can be put into use for evaluation and study by any one group of workers is small. This is unfortunate as we are thereby denied the opportunity to develop that insight into logical organization which can grow out of a working familiarity with many diverse forms. What is needed is a set of relatively simple, easily inter-connected modules from which working systems can be readily assembled for evaluation and study. With such a set, both the designer and user would be able to try out potentially powerful and novel structures on a very large scale, adjusting and improving the systems as needed. Once a design has been realized and its value established, it could then be reworked into tighter engineering form for maximum efficiency and for production by automatic wiring and fabrication techniques, and the experimental units made available for further studies or returned to "inventory" in the manner proposed by Estrin.[1]

The approach presented in the "following", papers describes modules which are primarily vehicles for experimental use and as such meet a set of requirements heretofore unnecessary in digital modules. Logical flexibility and ease of use are considered of primary importance while factors such as operating speed, economy, etc., are considered of secondary importance. The requirements can be summarized as follows:

(1) The modules must be functionally large enough to reduce logical detail by a significant amount and must be relatively easy to understand and assemble. The number of different types should be as small as possible so as to limit inventory, but at the same time, the set must be logically complete so that whole systems can be assembled. There must be not only central processor modules such as register and memory units, but also modules for power, signal conditioning, input-output buffering and control, together with a reasonable selection of input-output devices themselves.

(2) The mode of combining units into larger structures must be very simple (a problem first considered by Babbage, who examined this matter "with unceasing anxiety" one hundred and twenty years ago).[2] The modules should be designed for easy mechanical assembly. Communication from one mechanical assemblage to another should be accomplished by means of easily connected cables.

(3) All units should be designed so that the assembling of these units into a working system presents no logically irrelevant problems such as those relating to circuit loading, waveform deterioration, signal propagation delay, power supply interactions, and so forth, *regardless of the size or complexity of the system.* The modules should be powered and perhaps controlled individually, and all possible signal paths must be provided with signal-standardizing amplifiers capable of driving all possible loads.

We call units which meet these requirements *macromodules* to distinguish them from the more

*This research was supported in part by the Advanced Research Projects Agency of the Department of Defense through contract SD-302 and by the Division of Research Facilities and Resources of the National Institutes of Health through grant FR-00218.

conventional computer system modules. In this report we present a set of macromodules which, although not "complete" in the above sense, meets all other requirements and is sufficient for the synthesis of all central processor functions of which we are presently aware.

The following papers present the principal direction, achievements, and goals of the macromodular computer development program under way at Washington University. They describe an approach in which flexibility and simplicity are brought to the forefront of factors relevant in computer design. Two aspects of this approach are distinguishable.

First, it adds an experimental element to the theoretical and simulational techniques now available to the system designer. Perhaps this can most effectively be employed in research groups having as their main objectives the creation of advanced computer systems for particular areas of work. Because of the relative ease with which a given configuration of macromodules can be profoundly altered, it is possible for such a group to work actively with several widely different forms in an attempt to find optimal configurations for different problem classes.

Second, it makes possible a smoothness of growth and refinement in an operating computing system. Because of the electronic independence of the macromodules, it is relatively easy to expand a macromodular system and to add new functions without seriously affecting the continuity of on-going work and without jeopardizing any existing investment in programs and operating procedures.

These properties of macromodular systems are of growing importance as we turn increasingly toward the user for new concepts in the search for more effective information processing systems.

REFERENCES

1 G. ESTRIN
Organization of computer systems the fixed plus variable structure computer
Proc. WJCC 33-40 1960

2 L. F. MENABREA
Statement of the circumstances attending the invention and construction of Mr. Babbage's calculating engines
Philosophical Magazine 235 September 1848

A functional description of macromodules*

by SEVERO M. ORNSTEIN, MISHELL J. STUCKI and WESLEY A. CLARK
Washington University
St. Louis, Missouri

*This research was supported in part by the Advanced Research Projects Agency of the Department of Defense through contract SD-302 and by the Division of Research Facilities and Resources of the National Institutes of Health through grant FR-00218.

INTRODUCTION

This paper describes a set of macromodular building blocks such as registers, adders, memories, control devices, etc., from which it is possible for the electronically-naive to construct arbitrarily large and complex computers that work. Machines are assembled by plugging the modules into cells of a special frame which provides for communication between adjacent cells. Explicit data pathways and control structures are then made by plugging in standardized cables. All pieces of a system are therefore recoverable and systems can be reconfigured easily. Data modules process twelve-bit word-segments; greater word lengths are obtained by interconnecting modules. Memory modules hold 4096 twelve-bit segments and can also be interconnected to form larger arrays. Particular attention is given to the problem of designing control structures. The control signals for a given process are routed along the cables of a control network whose topology is isomorphic to the flow diagram representing the process. The step from conception to realization can therefore be made directly.

The task of defining a set of macromodules or building blocks is not unlike that of defining an instruction repertoire for a computer. The fundamental requirement is that the set be sufficiently general to permit construction of any central processor.* The particular set described here embodies this generality and satisfies the requirements set forth in another paper.[1] While it illustrates the approach we have taken, the set is by no means unique and, like a particular instruction code, will be more convenient or efficient for some tasks than it will for others.

General characteristics

The macromodules to be described are relatively small, dimensionally modular boxes which plug into a cellular frame structure, some modules occupying more than one cell. Each module contains all of the electronic circuits and memory elements required in the performance of its particular function. Connectors on the frame provide for communication between modules in neighboring cells, and assemblages of units are thus made by plugging them into appropriately adjacent positions. Faceplates attached to the modules' front surfaces provide the electrical connectors for signal access, and standardized cables are provided for inter-assemblage communication. All connectors are backed by signal-standardizing amplifiers capable of driving any adjacent module or attachable cable. Since all cabling takes place between faceplates which are separable from the modules, it is possible to remove modules from a frame for temporary use elsewhere without disturbing the cabling.

Data processing modules are organized in parallel binary form with a word-length modulus of 12 bits, and are designed functionally for asynchronous operation. Memory modules hold 4096 12 bit words.†

The design of a system based on these modules requires, we believe, only the exercise of logic. The operability of the resulting system is not critically affected by the physical distribution or arrangement of

*No input-output macromodules are discussed here although modules for various devices (scopes, tapes, printers, readers, etc.) are obviously required to complete the set.

†The numbers 12, 4096, and other such parameters have been made specific, for purposes of this paper, only to simplify description.

parts, the distances between units, the number or diversity of modules, or the routing of the interconnecting pathways. Macromodular systems are, as a result, capable of continuous growth and functional enrichment.

System organization

Macromodular systems may be viewed in terms of two logically distinct, interacting networks as shown in Figure 1. The *processing network* (the heavy-lined structure) consists of data processing elements interconnected by data pathways, and provides for the storage, propagation, and transformation of data within a system. The *sequencing network* (the light-lined structure) consists of control nodes distributed throughout the system, interconnected by control pathways. The structure of the processing network defines the basic data processing operations of the system while the structure of the sequencing network defines the order in which sub^eets of these basic operations can be carried out.

Interaction between these networks takes place at control terminals on the data processing elements. These terminals have two functions:)1) they allow the sequencing network to initiate operations, and (2) they return completion signals when the operations are finished. Each basic data processing operation has an associated set of these terminals (Figure 2), the number of terminals in the set being determined by the nature of the operation. Operations that manipulate data, *data operations*, have two, an initiation terminal and a completion terminal. Operations that check data for specific values, *decision operations*, have more than two, one of which initiates the operation while the others (completion terminals) indicate the value of the data found. Also shown in Figure 2 is a *time continuous transformation* element. This element, unlike those already described, performs its operation continuously. The data presented at its output changes directly in response to changes of input data rather than in response to control signals, and as a result, the element has no control terminals at all. An operation is initiated when a control signal arrives at an initiation terminal. The operation is executed and finally a control signal issues from the completion terminal and travels to the next control node in the sequencing network.

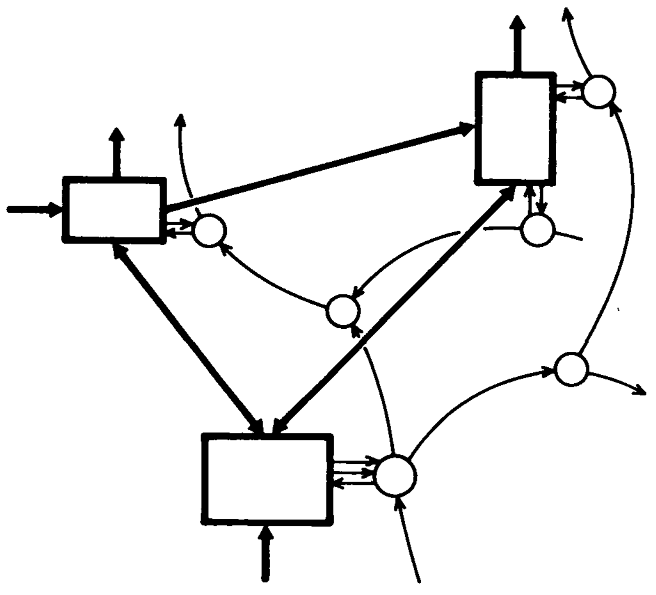


Figure 1 – Processing and sequencing networks

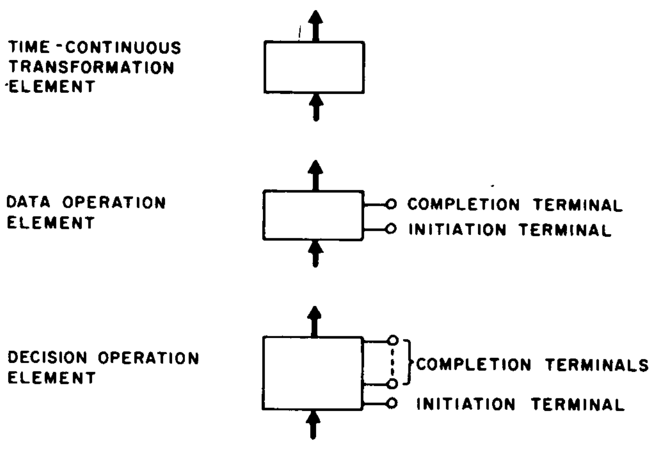


Figure 2 – Data processing operations

The order or sequence in which operations are performed is determined entirely by the structure of the sequencing network. This network is composed of signal nodes, calling elements, and interconnecting pathways. A *signal node* is an element which provides for the merging or branching of control signals

data operations, have two, an initiation terminal and a completion terminal. Operations that check data for specific values, *decision operations*, have more than two, one of which initiates the operation while the others (completion terminals) indicate the value of the data found. Also shown in Figure 2 is a *time continuous transformation* element. This element, unlike those already described, performs its operation continuously. The data presented at its output changes directly in response to changes of input data rather than in response to control signals, and as a result, the element has no control terminals at all.

The order or sequence in which operations are performed is determined entirely by the structure of the sequencing network. This network is composed of signal nodes, calling elements, and interconnecting pathways. A *signal node* is an element which provides for the merging or branching of control signals.
There are several types, two of which are shown in

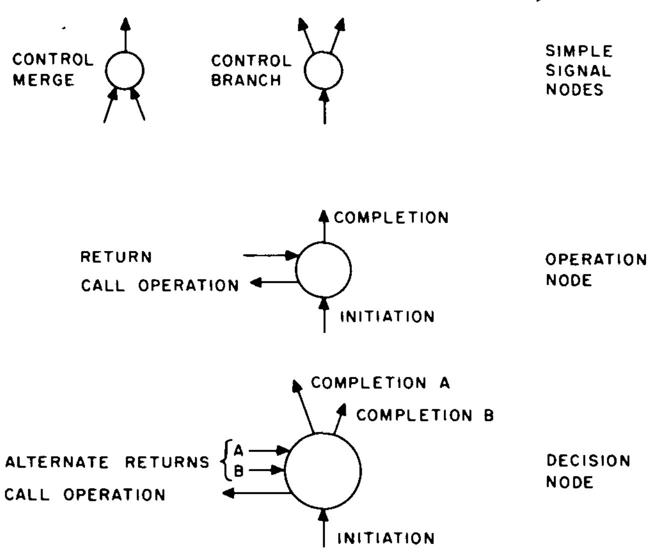


Figure 3 – Control nodes

Figure 3. A *calling element* is one which, when activated by a control signal at its initiation terminal, causes an operation to take place and, when signaled of completion of the operation, produces its own completion signal in turn. An *operation node* is a calling element for data operations, and a *decision node* is a calling element for decision operations.

Control within a macromodular system is asynchronous, that is, each event in a sequence of events can be initiated by the completion signal from the preceding event. The simplest way of accomplishing this is to connect a cable from the completion terminal associated with each operation to the initiation terminal associated with the next operation. This scheme, though simple and effective, has the limitation that once the control terminals for an operation have been connected for one sequence, it is no longer possible to incorporate the operation into any other sequence. In such cases, rather than connect to the terminals associated with the operation, we connect instead to the terminals of a calling element associated with the operation, as shown in Figure 4. Since any number of calling elements may call the same operation, an operation may thus occur in as many distinct sequences as necessary. Figure 4 illustrates this for two different sequences, namely, the sequence z, y, x and the sequence z, x. Since both sequences include the operations z and x, they initiate the operations through calling elements. Calling elements are not needed for operation y, however, as it appears in only one sequence and can therefore be incorporated by connections directly to its control terminals.

The control elements and interconnections defining a given sequence are said to be the *control path* for that sequence. The sequencing network is therefore the control path for the entire system.

Data validation

Whenever data values are used in either a data operation or a decision operation, it is necessary to be assured that 1) the results of all prior operations which could have perturbed the data are complete and 2) the new values of the data have propagated to the point of use regardless of the length of the pathway.

When information is used in the immediate locality of its source, i.e., within the same module, allowances for stabilization and signal progagation times are made within the module itself. When the source is remote from the point of use, a procedure known as *data validation* is followed to guarantee that the above two re-

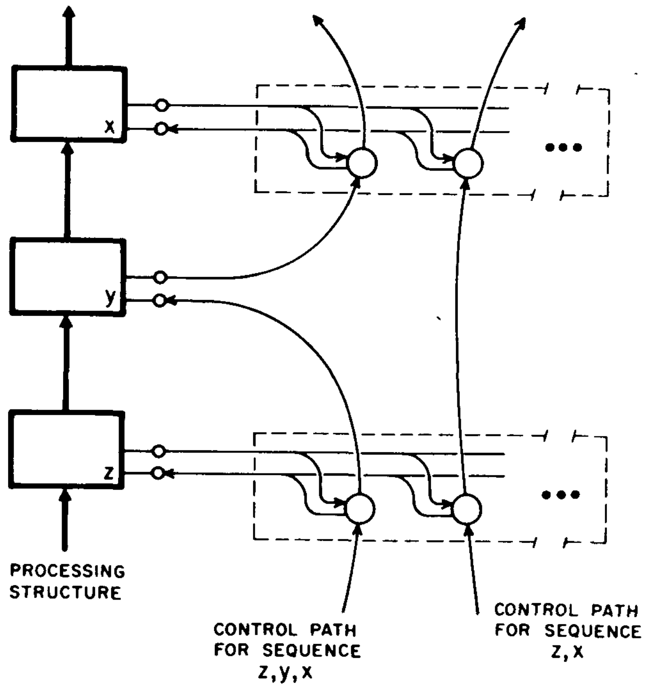

x
y
z
PROCESSING STRUCTURE
CONTROL PATH FOR SEQUENCE z,y,x
CONTROL PATH FOR SEQUENCE z,x

Figure 4 – Example of sequence control

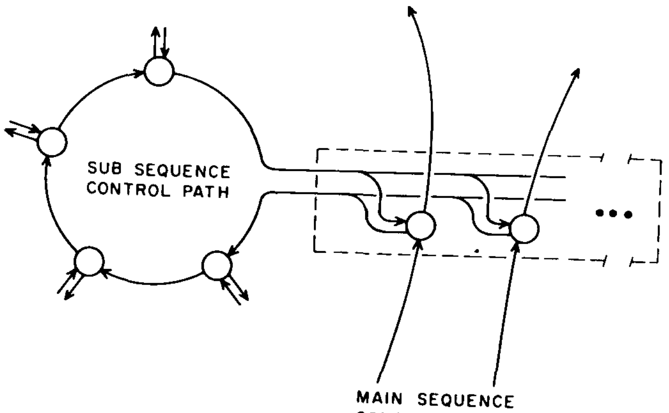


Figure 5 – Subsequence control

quirements are met. This process is discussed elsewhere.[2] For the present purposes, we shall assume that any attempt to use data following a perturbation of the data source will work properly, i.e. that the correct new value of the data will be used. Thus, there is no need for the designer to concern himself with details of propagation times so long as proper sequence is established.

The macromodules

We now proceed to give a functional description of the individual macromodules and illustrate their roles in various systems. Processing network elements are introduced first, and this is followed by a discussion of the various sequencing network elements. Power connections and the supporting frame structure are omitted from the figures to avoid obscuring the logical point being illustrated. A basic module type is sometimes fitted with more than one type of faceplate suiting it to different contexts. In such cases the circuits within the module sense the faceplate type and operation is suitably adjusted.

Cables

Data paths are constructed with *data cables*, control paths with *control cables*. These cables are made in a limited number of lengths, but cables of any length can be formed by using signal-standardizing extender units. A control cable contains a single channel for transmission of a control signal. A data cable contains 12 channels for the transmission of data and two for the transmission of signals associated with data validation. In the illustrations, data cables are drawn with heavy lines and control cables are drawn with thin lines.

Input switch sets

Any faceplate data input connector will accept either a data cable or an *input switch set*. The switch set is used to provide "constants" for presetting registers, masking, and so forth.

Registers

The basic register module Figure 6 contains a 12-bit register together with logic for the operations *clear, complement*, and *index* (count). Mounted on the faceplate of the register are control terminals for these operations together with a data output connector. Outputs are also carried up to the overlying cell. Data input to the register comes from units plugged into the underlying or overlying frame cells.

Registers of any length can be formed by plugging these modules into laterally adjacent cells in the frame (Figure 7). Only the faceplate for the right-

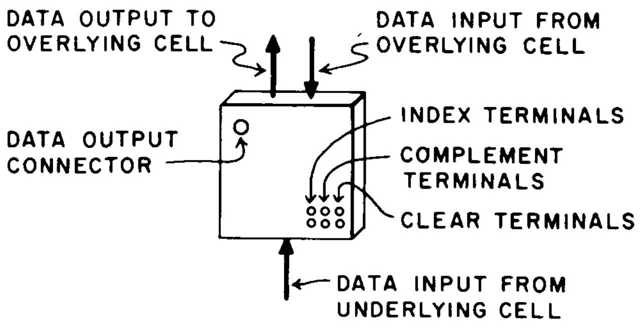

DATA OUTPUT TO OVERLYING CELL
DATA INPUT FROM OVERLYING CELL
INDEX TERMINALS
DATA OUTPUT CONNECTOR
COMPLEMENT TERMINALS
CLEAR TERMINALS
DATA INPUT FROM UNDERLYING CELL

Figure 6 – Register module

most module bears control terminals, and these terminals provide control for the whole register. This feature is common to all data operation and decision operation modules, thereby making control of an operation independent of register size. Special circuits within each module are coupled in such a way as to guarantee proper operation regardless of register length.[2]

Transfer operations

Data transfers from one register module to another require the use of a *data gate* module. This unit plugs into the frame cell underlying the receiving register and is connected by means of a data cable to the output of the data source register module (Figure 8). Twelve bits are transferred in parallel, and the transfer initiation and completion terminals appear on the data gate module. Transfers do not alter the information at the source.

If a register module is to receive input from n sources, n data gates are required (Figure 9). Stacking the units in this way allows each data gate to communicate with the receiving register module; any number of transfer paths into a register module can be provided. Two registers cannot exchange information without the aid of a third register, however, since simultaneity of events in different parts of an asynchronous system cannot be assumed.

Data gates plugged into laterally adjacent cells form *tiers* which provide for transfers into longer registers. Figure 10 illustrates a 24-bit transfer. Interconnect-

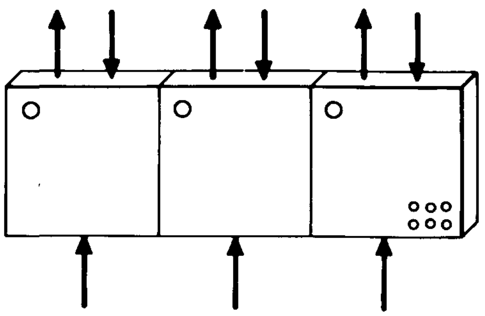


Figure 7 – Register extension

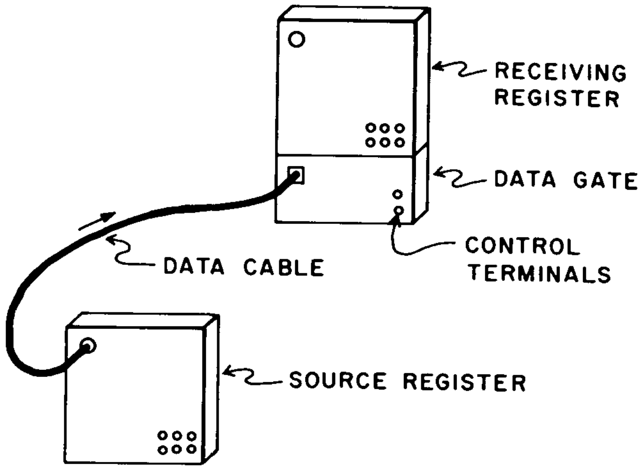


Figure 8 – Data transfer

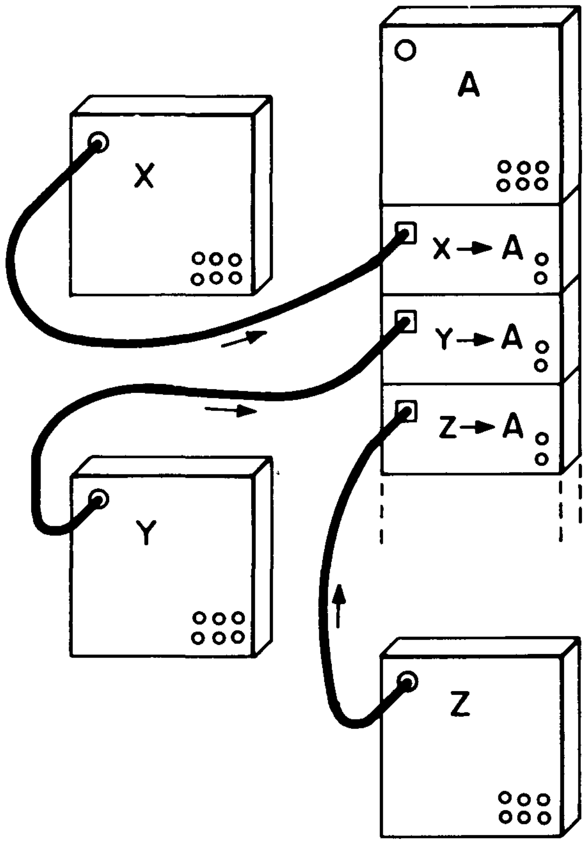


Figure 9 – Multiple transfers

ing data cables for the register segments may be of different length, as compensation for signal propagation times is automatically made in each cable.

In order to permit the transfer of information from a single source into more than one destination module, a *data branch* unit is used (Figure 11). Data branch units may be cascaded indefinitely to provide any number of connections to the same source. As it is not necessary to use both outputs, the data branch also doubles as an extender unit for data cables.

Memory

The memory module has a capacity of 4096 12-bit words and contains, in addition to a memory array, all required drivers, addressing logic, sense amplifiers, internal address and data registers, etc. Figure 12 shows a simple arrangement in which one memory module is used.

Reading is controlled by a pair of terminals on the module. In this example the memory module data output terminals are connected so that the word obtained from the memory array can be transferred into register B. The read operation's completion signal or any subsequent signal may be used to initiate this transfer operation.

Writing into the memory may take place from an arbitrary number of sources and for this purpose data gates are stacked in underlying cells just as for register transfers. Writing from each source is controlled by the terminals on the corresponding data gate.

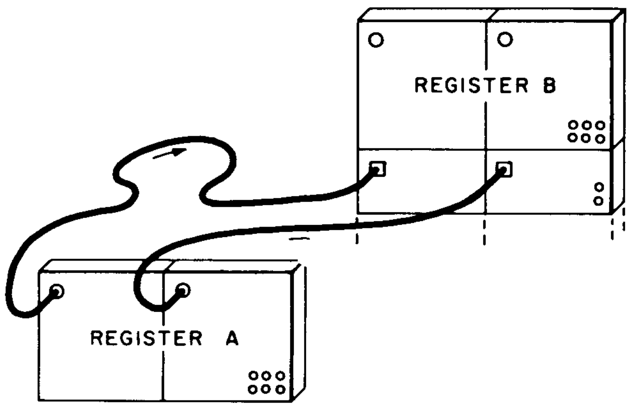


Figure 10 – Double length transfer

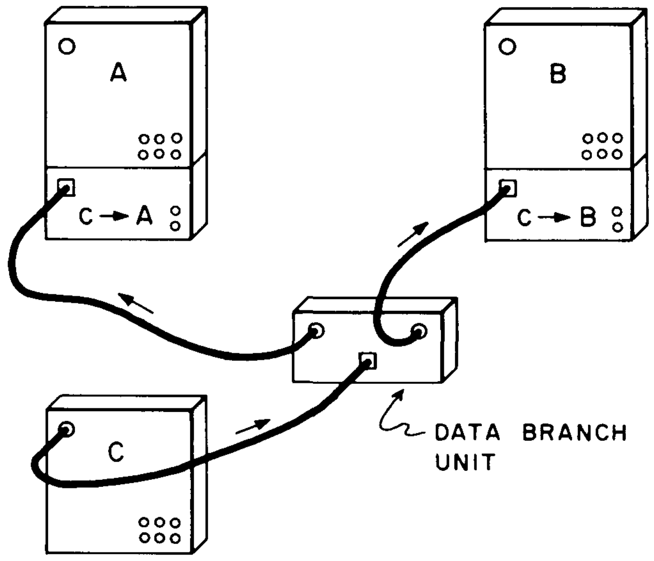


Figure 11 – Data branching

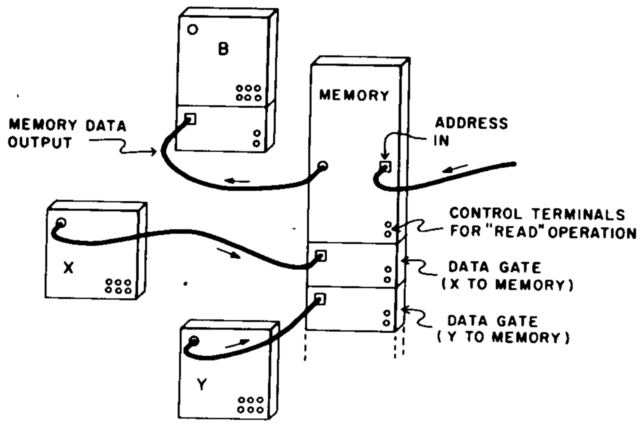


Figure 12—Simple memory

Memory modules can be plugged together laterally to increase word length and vertically to increase the number of words. Figure 13 shows a memory system containing 8192 thirty-six bit words. To permit referencing in memory systems containing more than 4096 words, connectors are provided for additional address inputs for selection of the appropriate tier. All access to the memory, regardless of address, is through terminals on the lowest tier.

Junction unit

A *junction unit* is a continuous transformer which permits one to rearrange bits within a word or to form words from bits selected from several words. The unit has two 12-bit data inputs and a single 12-bit data output. A set of jumpers can connect each of the 12 output terminals to any of the 24 input terminals or to fixed terminals supplying the value "1" or "0".

Suppose, for example, that information is to be transferred from parts of two 12-bit registers, A and B, into a third register, C. Specifically, suppose that the rightmost four bits of A are to be copied into the leftmost four bit positions of C, the lettmost four bits of B into the rightmost four bit positions of C, and the four middle bits of C are to be set to the binary value 1011. The units are arranged as shown in Figure 14, and the junction unit jumpers as shown in Figure 15. The word made up by the junction unit is transferred into C via a data gate.

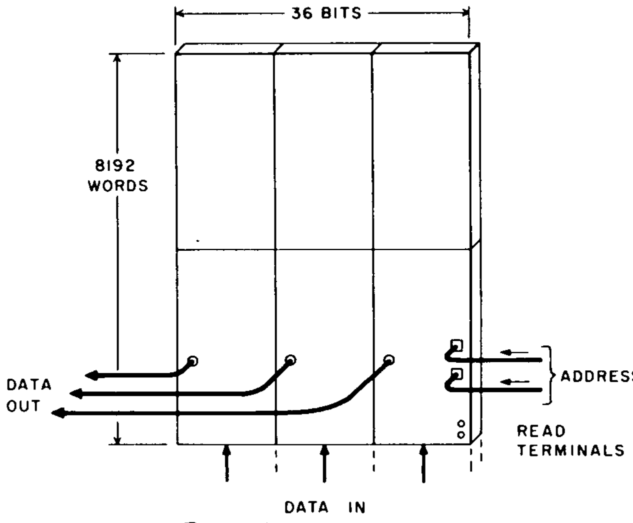


Figure 13—Memory extension

Data input from overlying units

Three types of units, namely, Shifters, Adders, and Function Units, overlie a register (or one another) and transmit their outputs only downward to the register via the implicit frame data pathway. These units have the following properties in common:

1 Each unit passes the data from the register on up to overlying cells and similarly, provides upward continuation of the down access route into the register.
2 Each unit operates in response to control signals presented to terminals on its faceplate.
3 Each unit uses the information originally held in the register below in determining the results to be returned to the register.
4 Each unit extends laterally as the register length extends, forming tiers which overlie the register

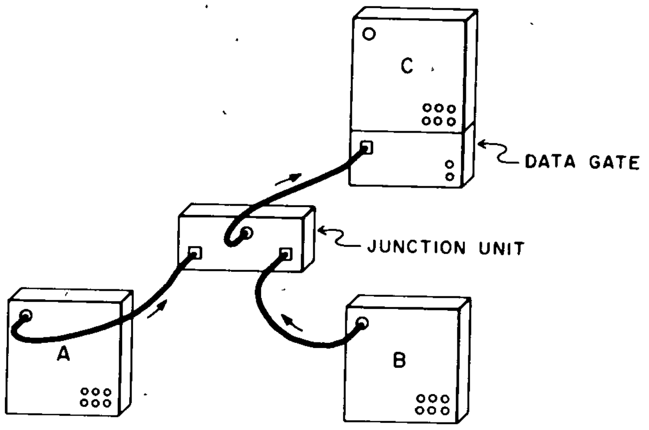


Figure 14—Junction unit

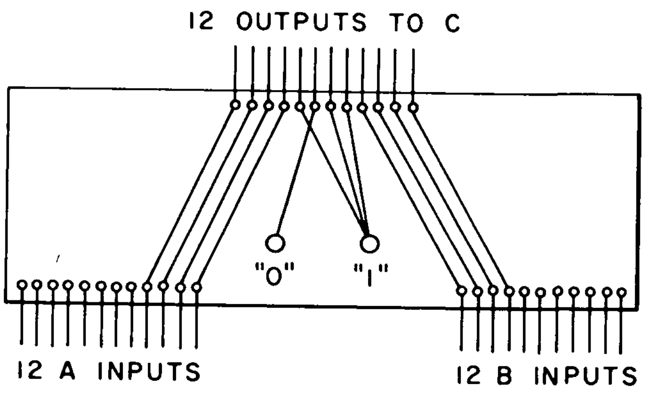


Figure 15—Junction unit jumpering

As many tiers as desired may be stacked on top of one another above the register. Each tier must extend across the full length of the register and must consist of only one type of unit. An example is shown in Figure 16.

Shift unit

Two types of shift modules are defined, one for shifting right and one for shifting left. Figure 17 shows a 36-bit register equipped to shift in either direction.

Each of five pairs of control terminals causes the register to shift one position, but each pair treats the incoming bit at the trailing end of the register differently. The options are as follows:

1 The bit is not changed.
2 The bit is replaced with the bit previously at the other end of the register, i. e. a rotation occurs.
3 The bit is set to the value "0".
4 The bit is set to the value "1".
5 The bit is taken from the data input port at the trailing end of the register. For leftward shifting the bit is taken from the most significant bit position of the input; for rightward shifting, from the least significant position. (The remaining 11 bits of information at the data input are not used.)

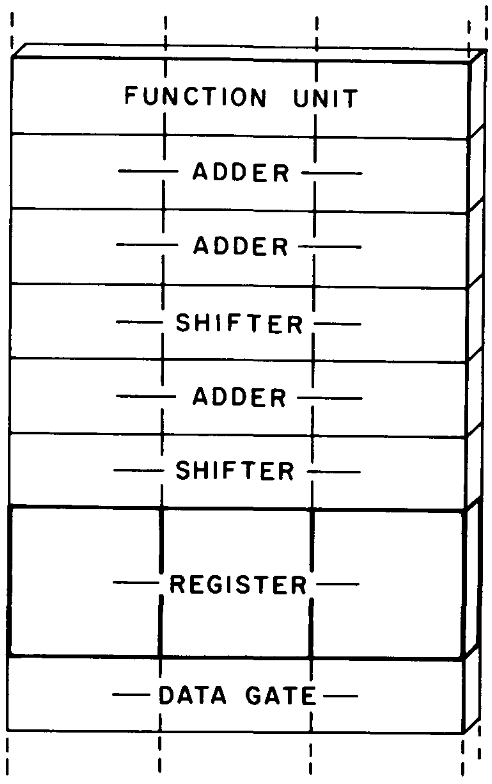


Figure 16 – Overlying units

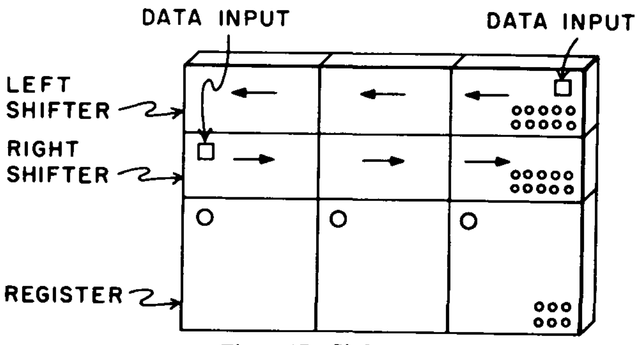


Figure 17 – Shift units

Adder unit

The adder unit takes one input from the underlying register and the other input from a data input port on the adder's faceplate. The sum is copied into the underlying register. Addition is controlled by terminals on the rightmost module in an adder tier. Figure 18 shows a 24-bit register, A, equipped to add from registers X or Y, (i.e. A + X→A or A + Y→A).

The adder unit contains three decision nodes which provide for the detection of overflow, negativity and the numerical zero sum. Their use is discussed below in the section dealing with control decisions.

Function unit

A function unit may perform any of three logical operations on a pair of data inputs. Like the adder, one of the data inputs comes from the underlying register and the other from a data input port on the front of the unit. The result of the operation is returned to the underlying register. The operations, controlled by three pairs of terminals on the rightmost unit, are the logical "OR" (v), the logical "AND" (·), and the "EXCLUSIVE OR" (v) Figure 19 shows a single 12-bit register equipped with a function unit as well as an adder.

An example of the use of the function unit might be the clearing of selected bits of the register. An input switch set may be inserted in the data input port of the function unit. If the switches are set to the value 1777_8, then whenever the "AND" terminals

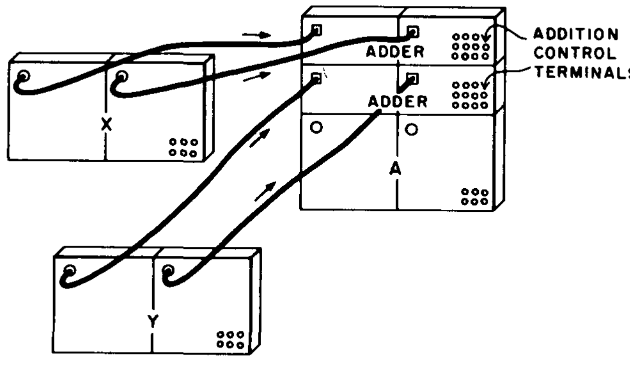


Figure 18 – Adder units

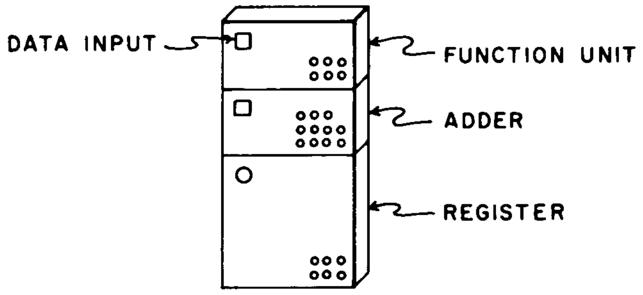


Figure 19 – Function unit

receive a control signal, the leftmost two bits of the register will be cleared.

Sequencing

In order to perform a desired sequence of operations, control signals are routed along control cables from one set of control terminals to the next set in a manner reminiscent of the plugboard programmed machines or the Bell Computer Model VI.[3] Thus, to perform the sequence

$$\begin{tikzcd} A \rightarrow B \arrow[d] \\ \text{INDEX } B \arrow[d] \\ B \rightarrow C \arrow[d] \\ {} \end{tikzcd}$$

one would interconnect control terminals as shown in Figure 20.

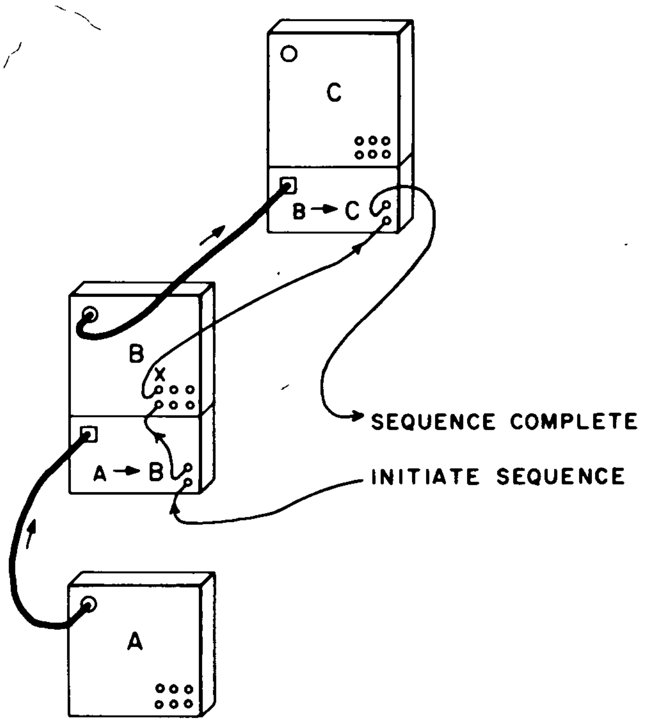


Figure 20 – Sequencing of operations

Concurrent sequences

Use of a control branch module (Figure 21) makes it possible to perform sequences of steps which may be executed concurrently. A control branch module contains several identical control branch elements. A control signal presented at the input terminal of such an element causes control signals to appear on each of two output terminals. These elements may be cascaded to form an arbitrary number of control path branches. They may also, of course, be used simply to extend control cables.

In the execution of two concurrent sequences, there will be found a point at which ensuing steps can be taken only after all steps of both sequences have been completed up to that point. A rendezvous element (Figure 22) which produces a signal (Z) at its output terminal only after signals have arrived at both of its input terminals, (X and Y) is used at the point of conjunction. Like the control branch, several elements are housed in a rendezvous module.*

For example, suppose a problem requires several set-up steps, one of which transfers data from register X to register A, and another of which transfers data from register Y to register B. These steps may either be executed sequentially (Figure 23), or they mav be executed concurrently (Figure 24). In the latter case, both transfers are activated and can take place at more or less the same time. As each

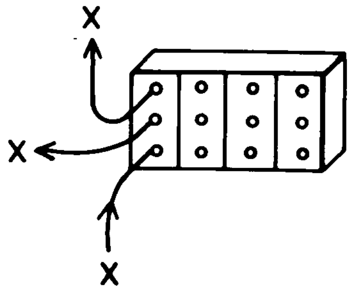


Figure 21 – Control branch module

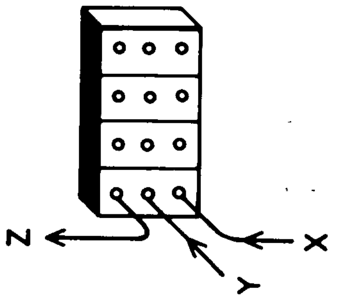


Figure 22 – Rendezvous module

*The rendezvous module is shown with a darkened top to distinguish it from units of a similar appearance.

transfer is completed, a signal is sent to the rendezvous unit. When both signals have arrived, the unit sends out a signal which proceeds on to the next step. A signal indicating the completion of an arbitrary number of concurrent actions can be generated by cascading rendezvous units.

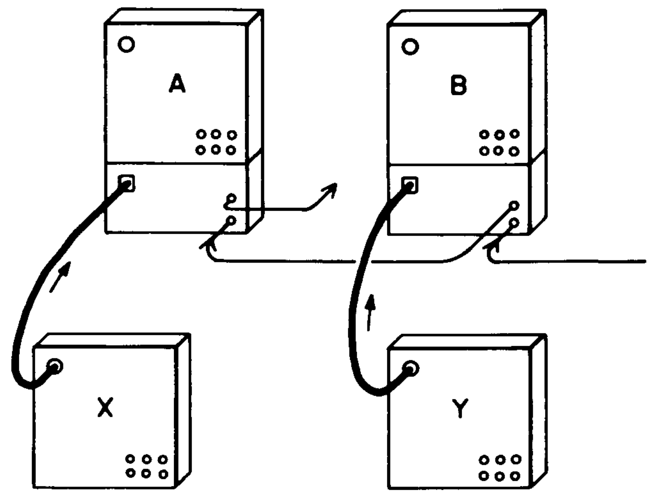


Figure 23 – Sequential transfers

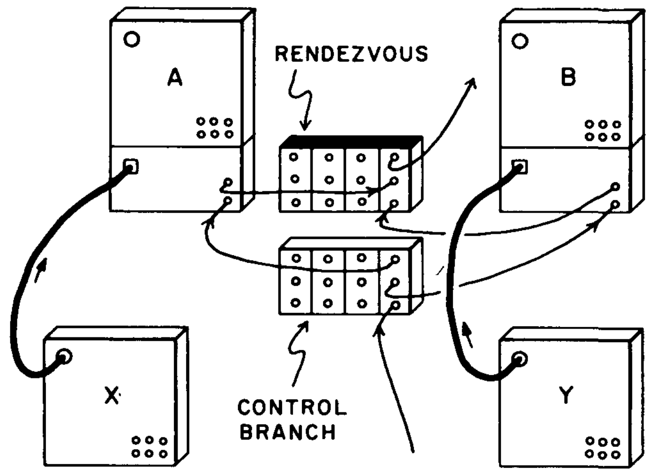


Figure 24 – Parallel transfers

Call unit

For situations in which more than one control path must have access to a single pair of control terminals, calling elements are used. Four calling elements are included in a single call unit.

A call unit is provided with the terminals shown in Figure 25. Terminals 1, 2, 3, and 4 are input terminals for the elements, 7, 8, 9, and 10 are output terminals. Whenever a control signal arrives at the input terminal of one of the calling elements, a control signal is presented at terminal 5 which thus initiates the operation. When the completion signal from the operation is returned to terminal 6, the call unit, in turn, produces a

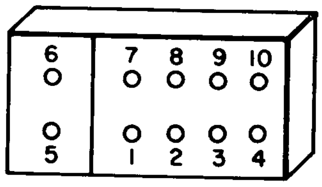


Figure 25 – Call unit

completion signal at the output terminal of the particular element which called for the operation. Call units may be attached to the control terminals for any operation. Figure 26 shows a call unit connected to the complement control terminals of a register module.

Call units may be cascaded, as shown in Figure 27, to increase the number of accesses for a particular operation to any desired number. Figure 28 shows three control paths, two of which contain a step which transfers A to B and two of which index A. The three sequences performed are:

(1)	(2)	(3)
COMPLEMENT A	*INDEX A*	*INDEX A*
A→B	A→B	
COMPLEMENT B	B→B	

No call units are required for the complement A, complement B, or B→C operation terminals, as each of these operations occurs in only one of the above sequences.

Sub – sequence calling

Call elements may be used to execute a sub-sequence common to several main sequences (for example, an operand fetch sub-sequence common to several instructions). After completion of the steps of the sub-sequence, each main sequence must continue with its own set of steps (corresponding perhaps to different instruction steps).

Figure 29 indicates, on the left, the steps of the sub-sequences S_1,S_2, and S_3. All of the instructions signal

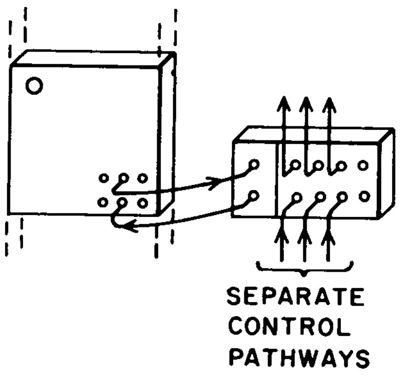


Figure 26 – A use of the call unit

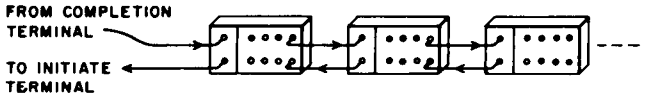


Figure 27 – Extension of call units

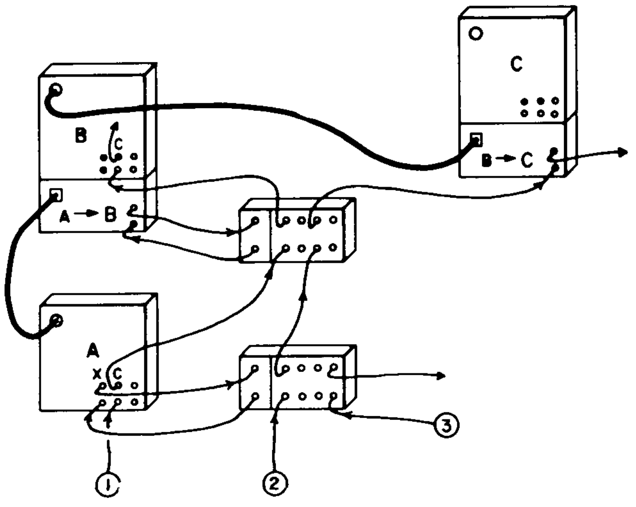


Figure 28 – Example of call unit use

the call unit assemblage on the right at the point in the sequence at which they require an operand. After all steps are completed, a signal is returned to the main calling assemblage, from which each instruction's control signal proceeds to initiate succeeding steps defining that particular instruction. Essentially, then, a call unit remembers which main control path is calling for the performance of a step or a set of steps during the execution of those steps.

Control decisions

In order to permit the choice of alternative steps to be made on the basis of data held by the system, two processing network elements, a detector unit and a decoder unit, are provided.

A detector unit is used to detect a specific value on a data path. It may be plugged into a cell overlying the register which provides data, or alternatively it may be connected via a data cable to a source of data. The binary value to be detected is entered in a set of 12 switches on the unit. A third setting of each switch allows one to indicate indifference to the value of the bit at that position.

A detector unit has three control terminals, one for interrogation and the others to indicate the result. When a control signal interrogates the detector unit, the data is compared with the pattern set in the switches. If the pattern matches the data, a control signal will be presented at the "Yes" terminal. If the pattern does not match the data, a control signal will be presented at the "No" terminal.

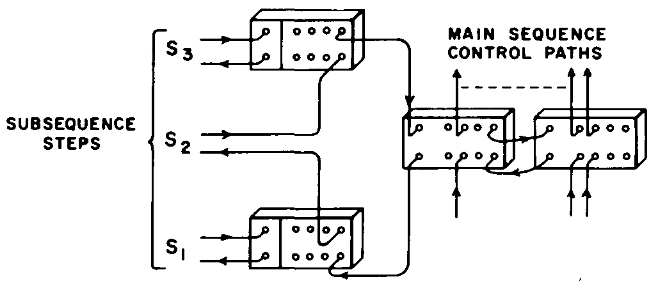


Figure 29 – Sub-sequence calling

Figure 30 shows a detector which tests for the value 101110 in the rightmost 6 bit positions of register A. X's indicate indifference to the leftmost 6 bits. In this configuration the detector unit receives its data input from the underlying register. Likewise, the detector unit passes the data on upward to the overlying cell. Detector units may thus be stacked one above the other in adjacent vertical cells (Figure 31) making it convenient to test for any of a variety of possible patterns of interest. Note that a shifter and an adder unit intervene between the register module and the detector stack. This is permissible in that these units also pass the register outputs upward. It is not permissible, however, to place an adder, shifter or function unit *above* a detector unit inasmuch as these units require downward access to the register for depositing their results, a feature not required by or incorporated in the detector unit.

Figure 32 shows the same stack of detectors, placed in cells not overlying the register. In this case, the source register outputs are delivered to the bottommost detector unit via a data cable.

Detector units extend laterally in the usual fashion for the detection of patterns of more than 12 bits. Figure 33 shows an arrangement which detects a pattern of 36 bits from a variety of sources.

The adder unit, as mentioned earlier, has three sets of detector terminals (Figure 33). These terminals are similar to control terminals on the detector unit and

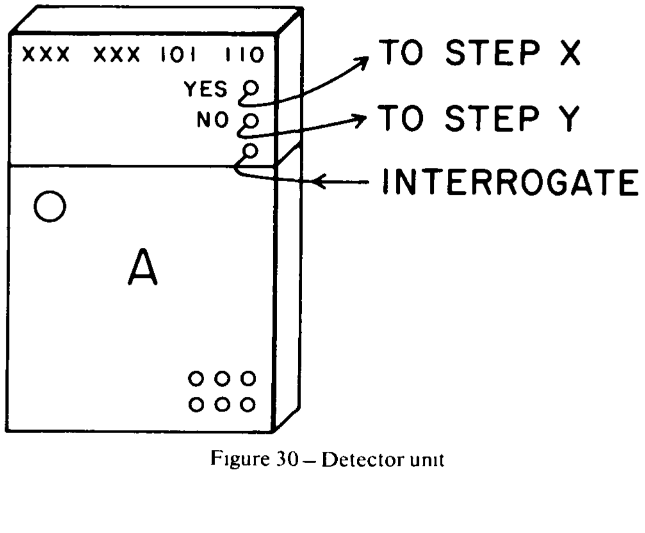


Figure 30 – Detector unit

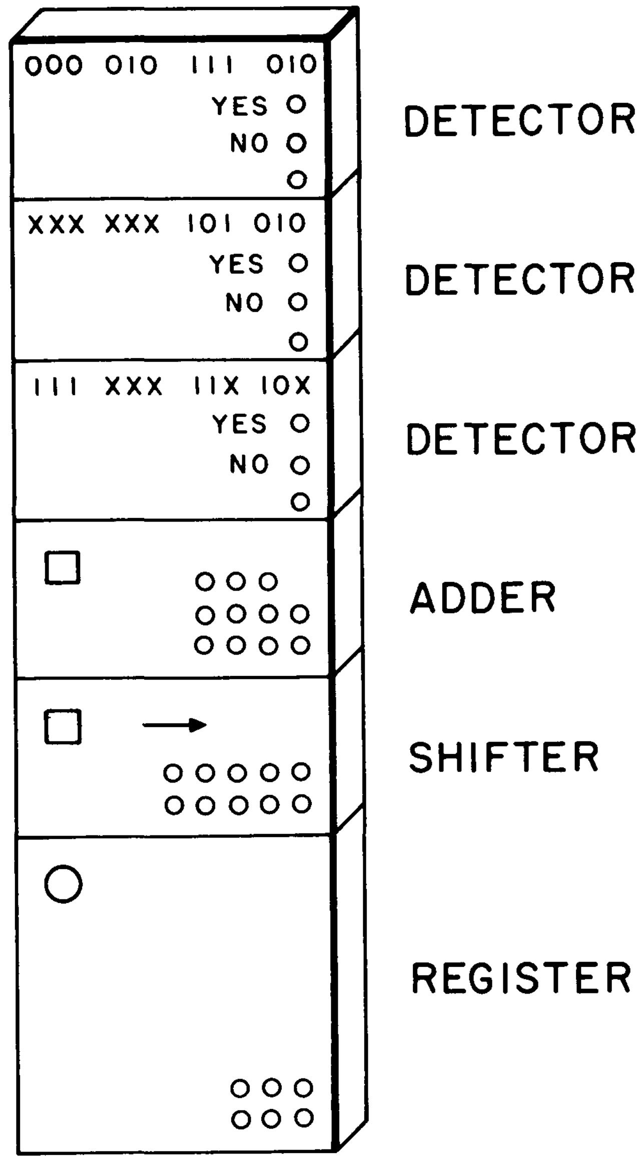


Figure 31 – Stacking of units above the register

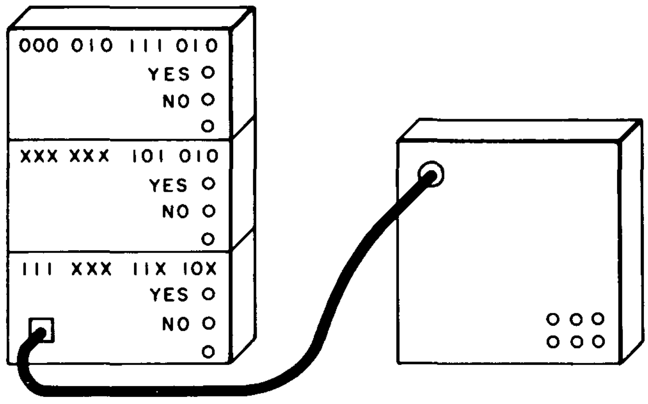


Figure 32 – Separate detector stack

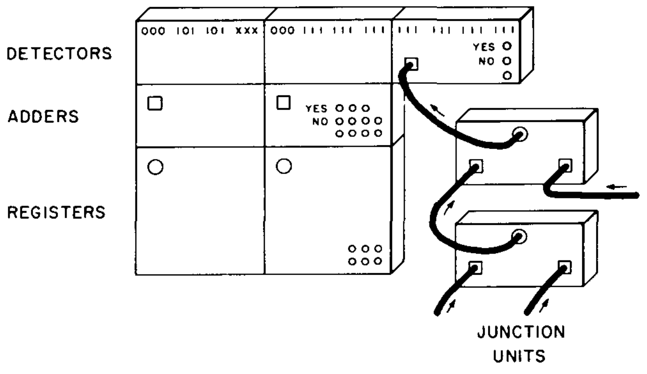


Figure 33 – Combined detectors

are used in an equivalent manner for detection of carry overflow, negativity, and the numerical sum zero. A signal returned at a "Yes" terminal indicates that the associated condition exists.

Detector units make it possible to select one of two alternative control paths on the basis of particular data values or patterns. Sometimes, however, it is desirable to select one of 2^n paths on the basis of n bits of data, and for such cases a decoder unit is provided (Figure 34). This unit contains a 3-bit decoder which may be interrogated by a control signal. Data input comes to the decoder either from the underlying cell or via a data cable. The input is passed upward to the overlying cell. Jumpers within a unit select three of the 12 data lines for decoding. When a control signal arrives at the interrogate terminal, a signal is produced on one of the eight output terminals. Each output terminal corresponds to one of the eight possible values encoded on the selected bits.

To permit decoding of values encoded on fewer bits, the jumpers for bit selection can provide an apparent "0" to the decoder. If, for example, the most significant bit of the decoded subset is thus fixed, an output signal will never appear on lines 4, 5, 6, or 7.

Figure 35 shows a stack of decoder units which splits the control path into one of 32 alternatives based upon bits 0-4 of the data input.

Merge unit

At some point after making a decision, all of the decision-dependent steps will have been executed, and the corresponding alternate control paths may be joined through the use of a merge element, several elements being housed in a merge unit (Figure 36). A merge element produces a signal (Z) at its output terminal whenever a signal appears at either input terminal (X or Y). Cascading permits the merging

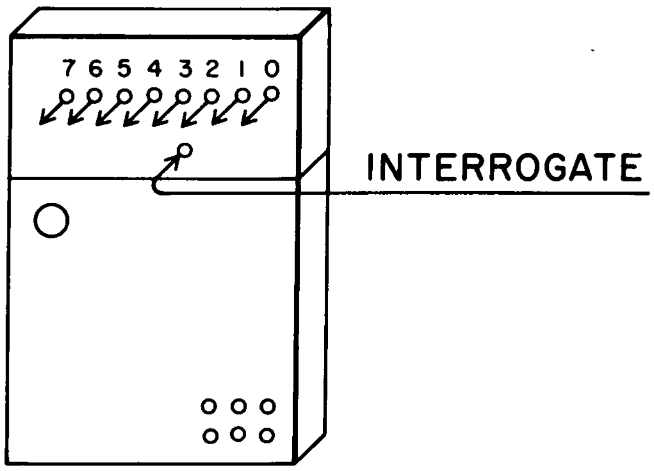


Figure 34 – A decoder unit on a register

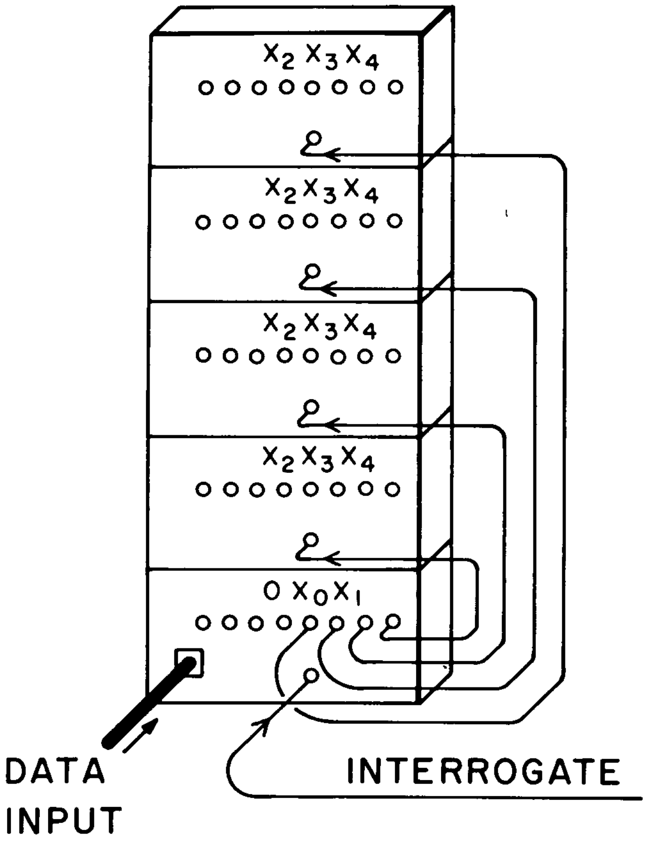


Figure 35 – Decoding five bits

of as many paths as desired. This unit, like the control branch, may also be used to interconnect, and thereby extend, control cables.

Decision call unit

A decision call unit permits a detector unit to be accessed by more than one control path and contains four decision calling elements as shown in Figure 37. This unit is connected via control cables to the control terminals of a detector unit, terminal 5 to the interrogate terminal and terminals 6 and 7 to the "No" and "Yes" terminals as shown in Figure 38. When a control signal is presented at the input of any of the four decision calling elements, a signal is produced at terminal 5 which interrogates the detector unit. A "Yes" or "No" signal is returned to the decision call unit and will appear at the "Yes" or "No" terminal of the element which called for the interrogation.

Decision call units can be cascaded (Figure 39) to allow an arbitrarily large number of control paths to access the same detector unit. Like the call unit, the decision call unit may be used to provide multiple access to a sub-sequence control path. In this case, the sub-sequence may include a decision in which one of two alternative control paths is selected.

Interlocking

In some situations two independent sequences will both require the use of the same data-processing element or elements (e.g., two sequences which make use of the same memory), and conflicts may arise. For such situations an interlock unit (Figure 40) is provided. This unit sorts incoming control signals on a "first-come, first-served" basis, interlocking them in such a way as to resolve conflicts.

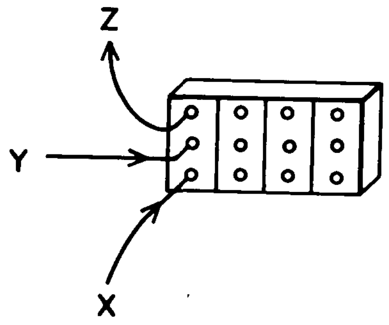


Figure 36 – Merge unit

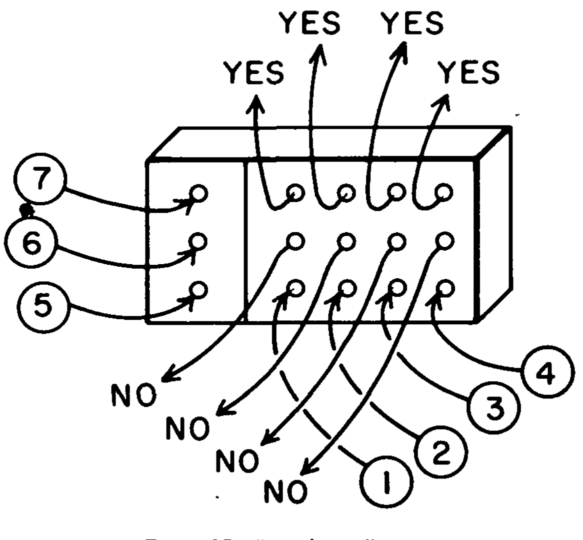


Figure 37 – Decision call unit

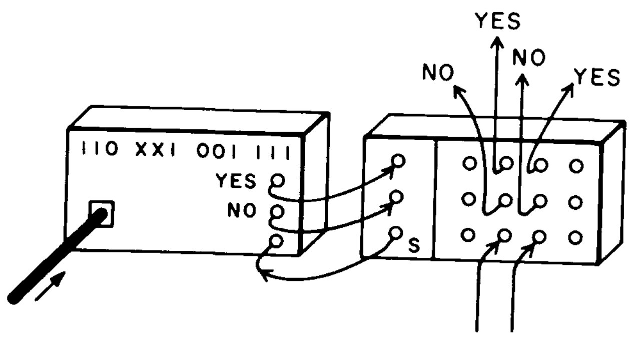


Figure 38 – Example of decision calling

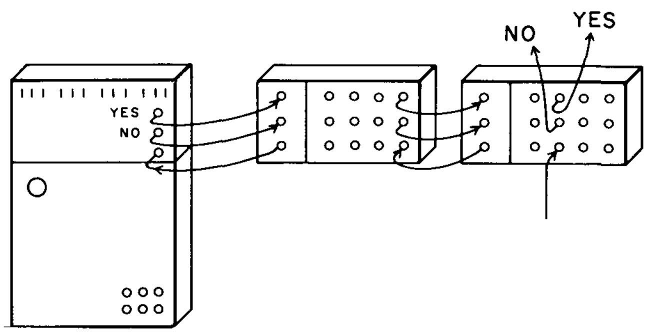


Figure 39 – Extending decision calls

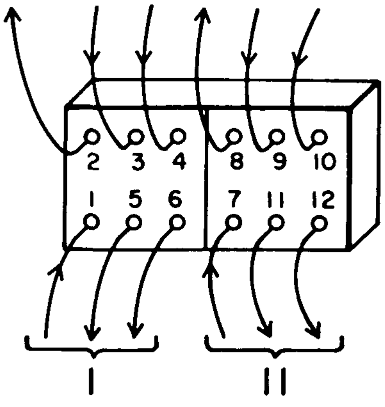


Figure 40 – Interlock unit

The left and right halves are associated with the control paths, I and II, of two concurrent sequences which must be interlocked. For a sequence to enter an interlocked phase, a signal must be presented to the interlock at an input terminal (terminal 1 for I, terminal 7 for II). Because it must be assumed that the sequences do not necessarily contain the same steps within their interlocked phases, each control path is provided with its own terminals (terminals 2, 3, 4 for I; terminals 8, 9, 10 for II) for use during their interlocked phase. If the interlock is off when a signal arrives at terminal 1, it is turned on and a signal is produced at terminal 2. This signal initiates the steps within the interlocked phase of the sequence associated with control path I. After the last of these steps has been completed, a signal is returned to terminal 3 or terminal 4. (Two return terminals are provided to allow for a possible decision within the interlocked section.) The return produces a signal at either terminal 5 or 6, depending on whether the return came to terminal 3 or 4, and shuts off the interlock. An equivalent process takes place for control path II, using terminals 7 through 12. If either control signal enters the interlock while it is on, it will be held up until the interlock is turned off. If signals arrive at terminals 1 and 7 simultaneously, only one will be accommodated immediately; the other will wait its turn.

Figure 41 shows an arrangement for interlocking two sequences (I & II), both of which use register A. Interlock units plugged into laterally adjacent cells (Figure 42) permit interlocking of any number of sequences.

Example of a small computer

Let us consider the central processing portion of a very simple computer and sketch out how it might be realized in macromodular form. The computer has a 12-bit word length, 4096 words of programmable memory, and an instruction repertoire consisting of

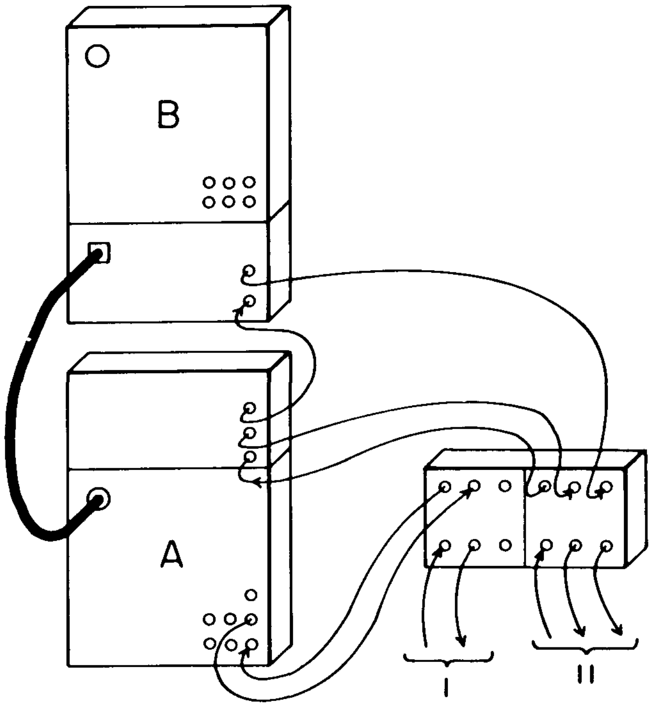


Figure 41 – Example of interlocking

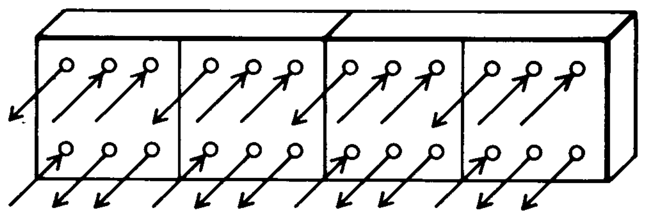


Figure 42 – Extended interlocking

eight instructions encoded on the leftmost three bits of the 12-bit instruction word.

Three of the instructions make reference to other memory locations by a process of indirect addressing. These instructions are ADDβ, STOβ, and JMPβ and use the contents of memory register $\beta(0 \leq \beta \leq 777_8)$ as the effective address. Thus, for example, if register β contains the number 1476, execution of an ADDβ instruction will cause the contents of register 1476 to be added to the contents of the accumulator. Similarly, a STOβ will store the contents of the accumulator in register 1476. A JMPβ will cause the next instruction to be taken from location 1476.

The remaining five instructions do not make reference to other memory locations. Instead, they perform the following functions:

CLR -	Clear the accumulator
COM -	Complement the contents of the accumulator.
APO -	Skip the next instruction if the accumulator contains a positive number.
SHL n -	Shift the contents of the accumulator n places to the left where n is specified by the four rightmost bits of the instruction word.
NOP -	Proceed immediately to the next instruction.

Figure 43 gives a two dimensional view of the processing network for this machine. Register A is the accumulator; register S is used to provide addresses to the memory and also to count out the number of shifts required by an SHL n instruction. During the execution of an instruction, register P holds the address of the next instruction to be executed. The decoder unit is set to decode the instruction field (3 most significant bits) of the word from memory, (M). The junction unit is set to mask out the instruction field when transferring β to S.

Figure 44 shows a flow diagram representation of the operation of the machine. Entering the top of the flow diagram corresponds to getting the next instruction from the memory. The contents of P are transferred to S in order to locate the instruction, and P is then indexed in preparation for locating the following instruction. Memory is read to obtain the instruction and the decoder is signaled to decode the instruction field of the memory word. Eight decoder output control paths are provided, one for each of the instructions, and a control signal is produced on the selected path. After the required sequence of steps has been executed, the control signal returns to get the next instruction, (GNI).

Figure 45 shows the control paths required to locate and decode the instructions, as well as the execution control paths for the instructions CLR, COM, APO, and NOP. The detector in the APO control path is set to detect a "0" in the leftmost (sign) position of the accumulator. A "No" signal response from the detector proceeds to the merge unit assemblage from which a signal to get the next instruction emerges. A "Yes" response indexes register P and then proceeds to get the next instruction. The NOP instruction has no execution steps, and its control path is routed directly to the GNI merge unit assembly.

The SHL n instruction, shown in Figure 46, uses the the S register to count the number of places which have been shifted. The SHL n instruction word is transferred from the memory into S and complemented. A detector on S monitors the rightmost four bits, (n), and provides a "Yes" response when the value 17_8 (the minus zero in ones' complement form) is detected. A control loop is entered in which the detector is interrogated. A "No" signal response from the detector shifts A one place, indexes the count in S, and again checks the detector. This process repeats as long as more shifts are required. When the required number of shifts has been completed, interrogation of the detector will produce a signal at its "Yes" output, which is routed to the GNI merge unit assemblage.

The control paths for the three memory reference instructions are routed to a sub-sequence all unit. This sub-sequence fetches the operand address and places it in the S register. This three-step sub-sequence is shown separately on the flow diagram in Figure 44 and the control path which realizes it is

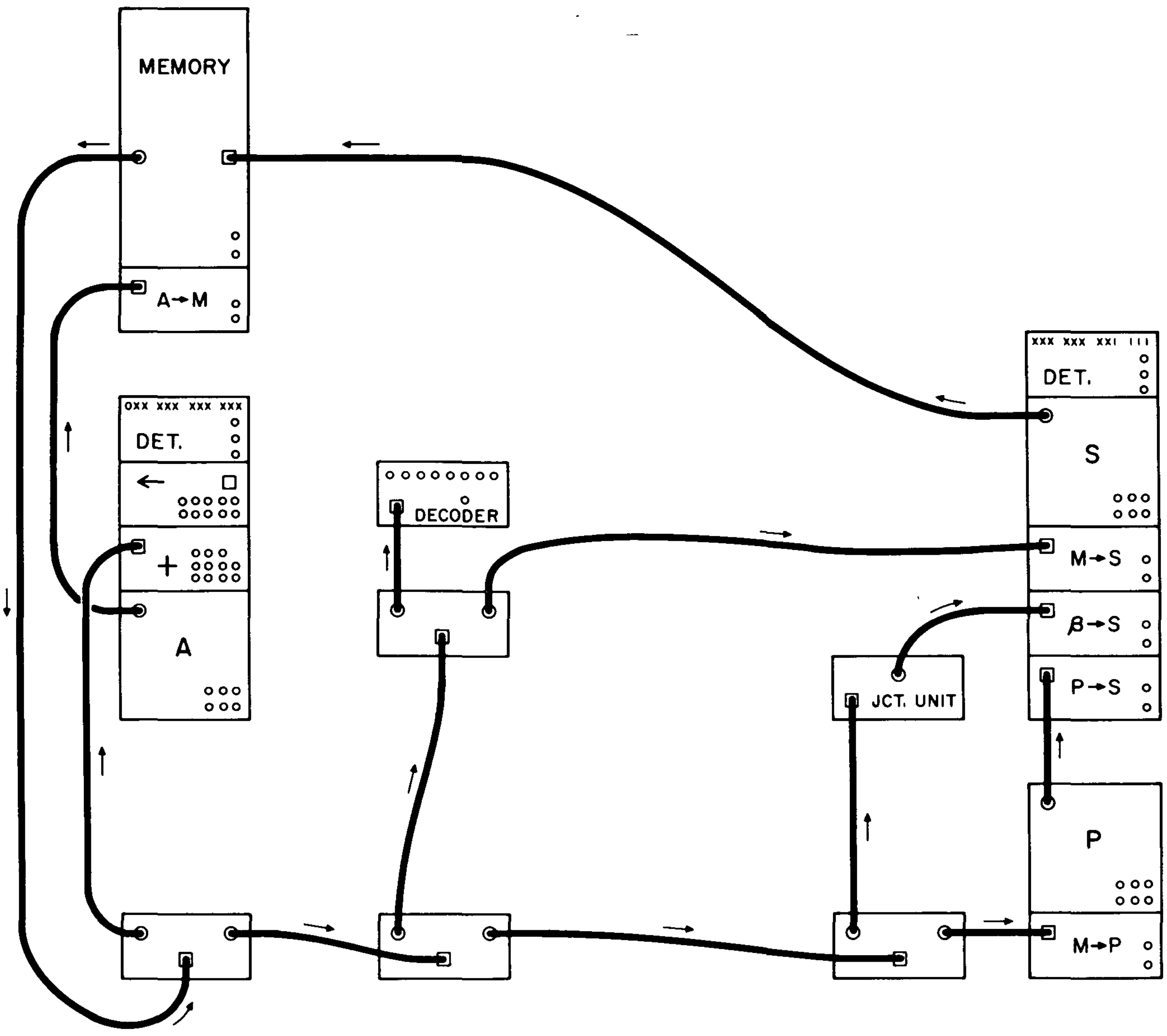


Figure 43 – Small computer processing network

shown in Figure 47. The first step of the sub-sequence is the transfer of β to S. In the following steps, the effective address is obtained from memory register β and transferred into S. Beyond this point the three instructions' control paths diverge.

In the ADD instruction the memory is again read to obtain the operand, which in the next step of the sequence, is added into A. The STO instruction execution sequence consists of a single Write command to the memory, as register A is connected directly to the memory data input. In the JMP instruction, the effective address is transferred into P from the memory. For this instruction the final step of the operand address fetch sub-sequence (which put the operand address into S) was unnecessary but harmless.

This example demonstrates the ease and directness with which simple systems can be put together. A total of 13 data cables, approximately 50 control cables, and 26 modules are required to form this complete, albeit comparatively modest, central processor.

CONCLUSION AND ACKNOWLEDGMENTS

The ideas presented here have evolved gradually through a combination of individual effort and group discussions. The authors wish to express their gratitude particularly to A. Anne, J. R. Cox, Y. H. Chuang,

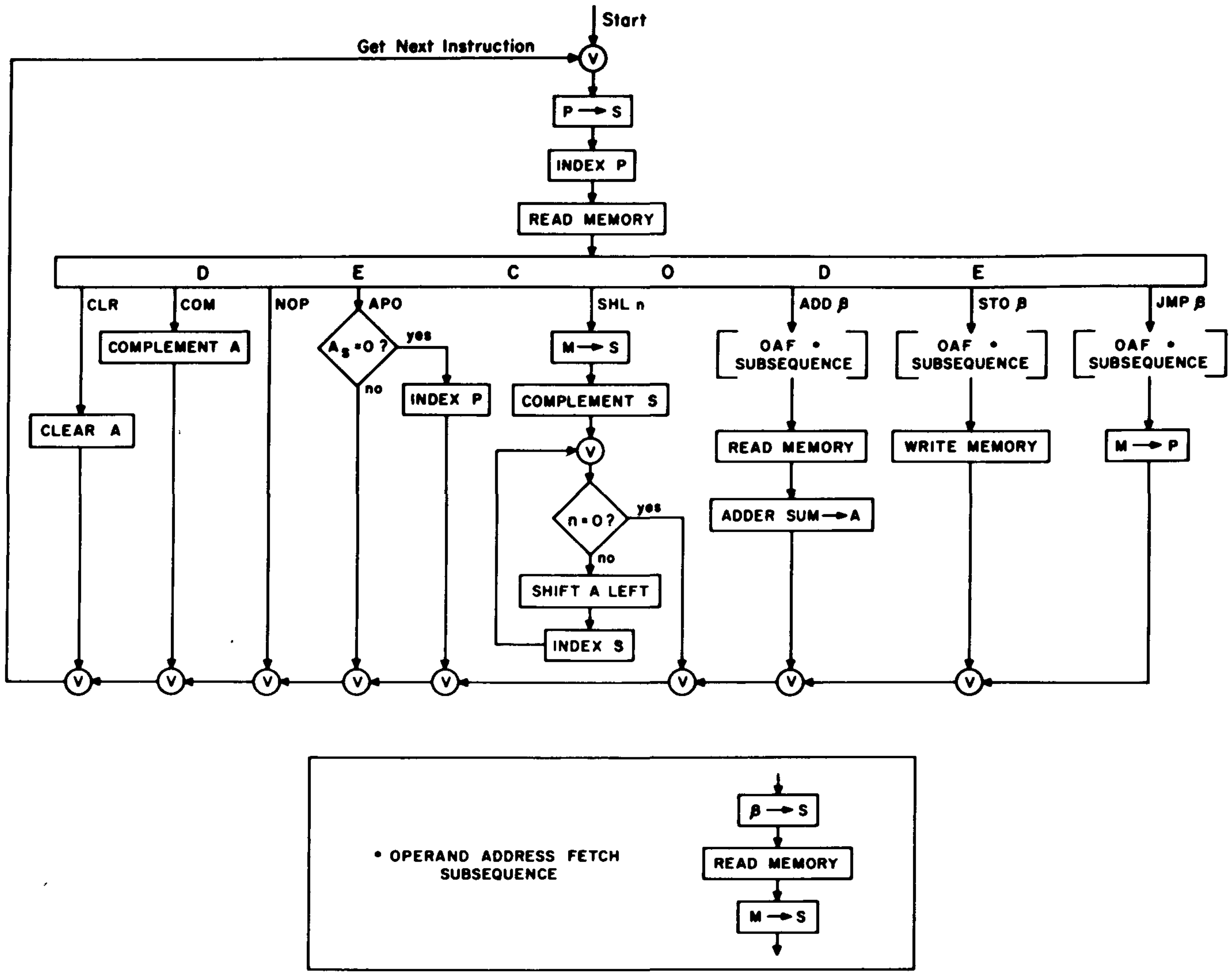


Figure 44 – Small computer control flow diagram

R. A. Ellis, G. C. Johns and C. E. Molnar who have contributed helpful criticism and suggestions.

While some details of design are still in flux, the die for the basic scheme is cast and an initial evaluation effort is under way. Prototypes of the macromodules are working and some small initial systems are planned for coming months. Paper design of these systems indicates that the particular functional breakdown we have chosen is a reasonable and convenient one.

Addition of new modules to the inventory together with some reshaping of those presently defined will certainly take place as experience guides us toward ever increasing convenience and flexibility.

REFERENCES

1 W A CLARK
Macromodular computer systems
Proc SJCC 1967

2 M J STUCKI S M ORNSTEIN and W A CLARK
Logical design of macromodules
Proc SJCC 1967

3 E G ANDREWS
The Bell computer model VI

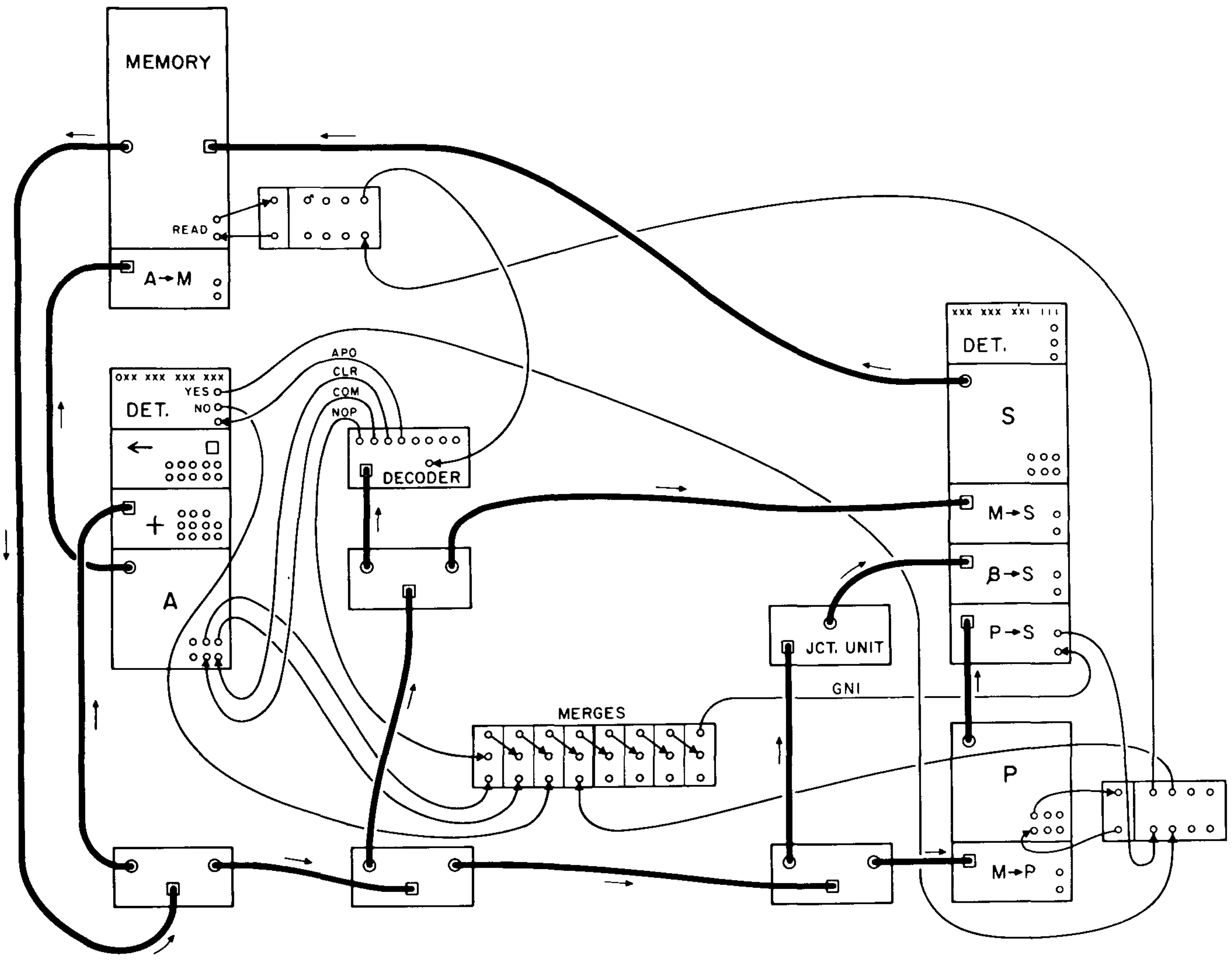


Figure 45 – Small computer-sequencing network for CLR(COM, APO, and NOP instructions

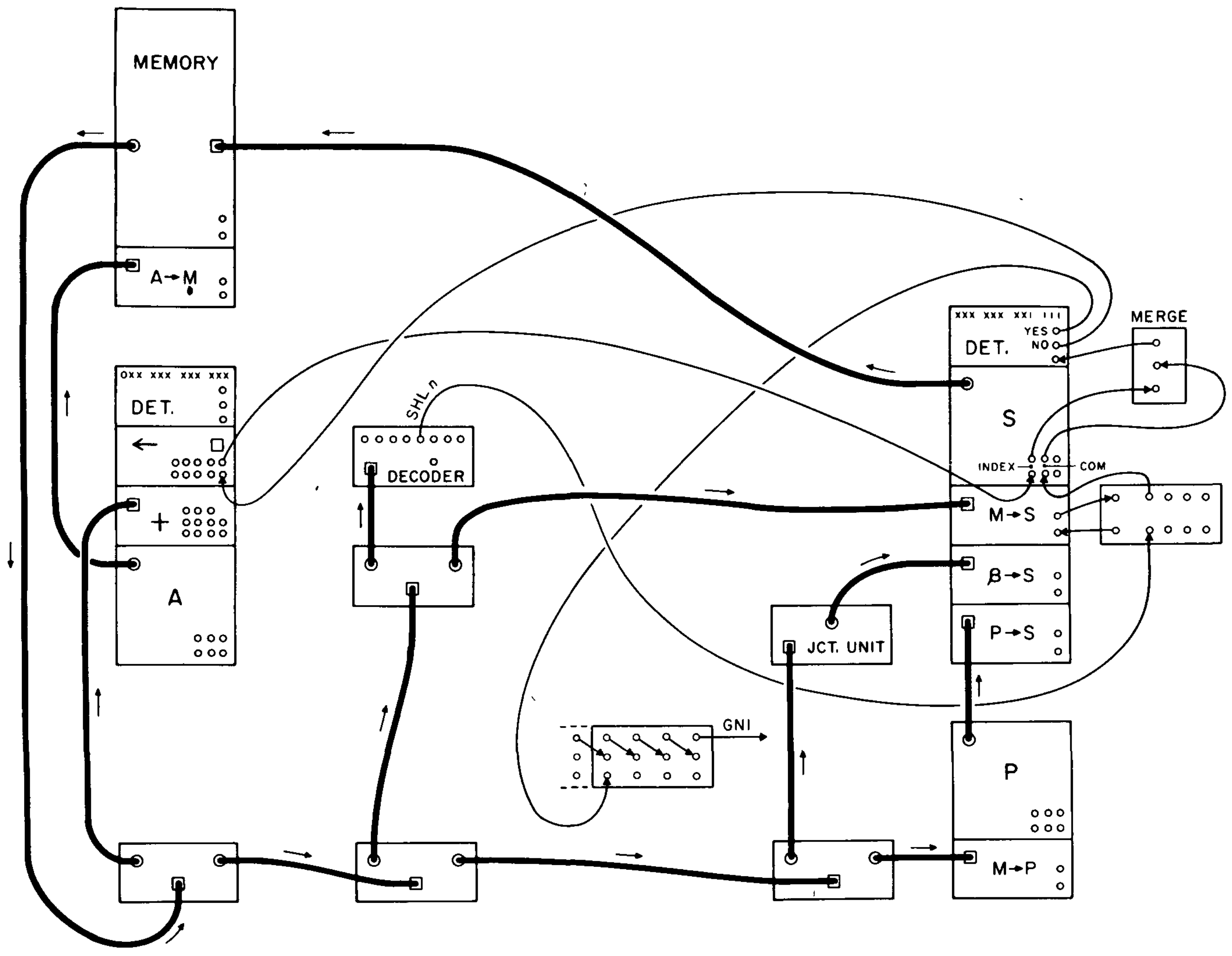


Figure 46 – Small computer-sequencing network for SHL n instruction

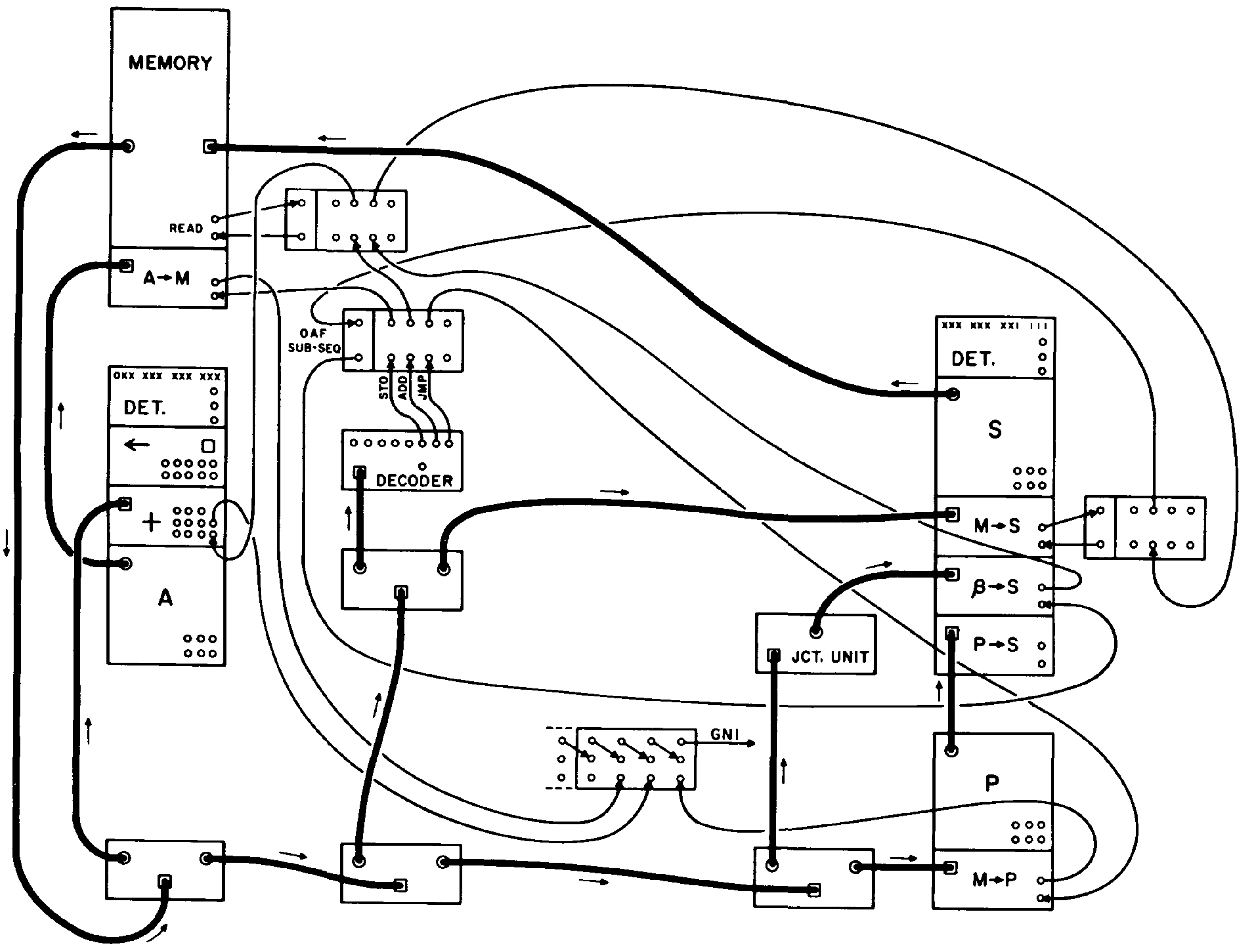


Figure 47 – Small computer-sequencing network for ADD, STO, and JMP instructions

Logical design of macromodules*

by MISHELL J. STUCKI, SEVERO M. ORNSTEIN and WESLEY A. CLARK
Washington University
St. Louis, Missouri

INTRODUCTION

The macromodules[1,2] being developed at Washington University are logical building blocks which can be inserted into a special frame and inter-connected by standardized cables to form digital computing systems of any desired complexity. The logical design of these modules is fraught with many problems, some of which yield easily to standard design techniques and others which do not. The purpose of this paper is to present the design approaches in present use for the handling of problems of the latter type. Specifically, rather than present the details of adders, shifters, registers, etc., discussion is confined to those aspects of the logic within the modules which simplifies the job of assembling the modules into a working system. The general areas of asynchronous control, data validation, and word-length extension are discussed and design approaches presented. These approaches are then illustrated in the design of the macromodular data transfer operation, and the paper concludes with a few general comments on the circuitry now in use.

Asynchronous control

One of the more interesting aspects of the macromodules is the general control scheme which allows complex system control structures to be implemented with relative ease. In this scheme, data processing modules are designed for asynchronous control and special control modules are provided for the paralleling and conditional branching of control signals. By "asynchronous control" it is meant that associated with each data processing operation that a module can perform is an initiation terminal and a completion terminal; execution of an operation begins when a control signal arrives at the initiation terminal, and at the conclusion of the operation, a control signal is generated at the completion terminal. The design of circuitry exhibiting this kind of behavior is one of the more difficult logical design problems. The approach used is to partition the circuitry for a given data processing operation (such as addition) into less complex asynchronous circuits and to regulate their behavior with a control network that utilizes the basic circuits of the control macromodules. The basic control circuits and their usage in control networks are described in the following paragraphs.

The circuitry associated with an operation designed for asynchronous control is represented diagrammatically by a circle containing the name of the operation (Figure 1). The incident and extant arrows

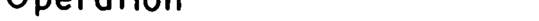


Figure 1 — Diagrammatic representation for the circuitry associated with an operation designed for asynchronous control

indicate connections to the initiation and completion terminals of the circuit. Operations can be made to occur in a specific sequence by connecting the completion terminal of each operation to the initiation terminal of the succeeding operation. Figure 2, for example, shows the flow diagram and connection network for the sequence a_1-a_2-a_3. The arrows in Figure 2b indicate that the completion terminal for a_1 is connected to the initiation terminal for a_2 and the completion terminal for a_2 is connected to the initiation terminal for a_3. Connected in this way, the completion signal from each operation initiates the next operation in the sequence.

*This research was supported in part by the Advanced Research Projects Agency of the Department of Defense through contract SD-302 and by the Division of Research Facilities and Resources of the National Institutes of Health through grant FR-00218

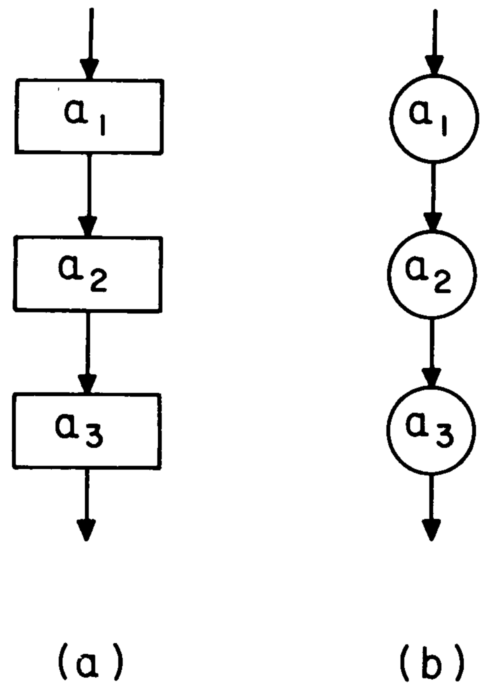


Figure 2 — Flow diagram (a) and control network (b) for a simple sequence

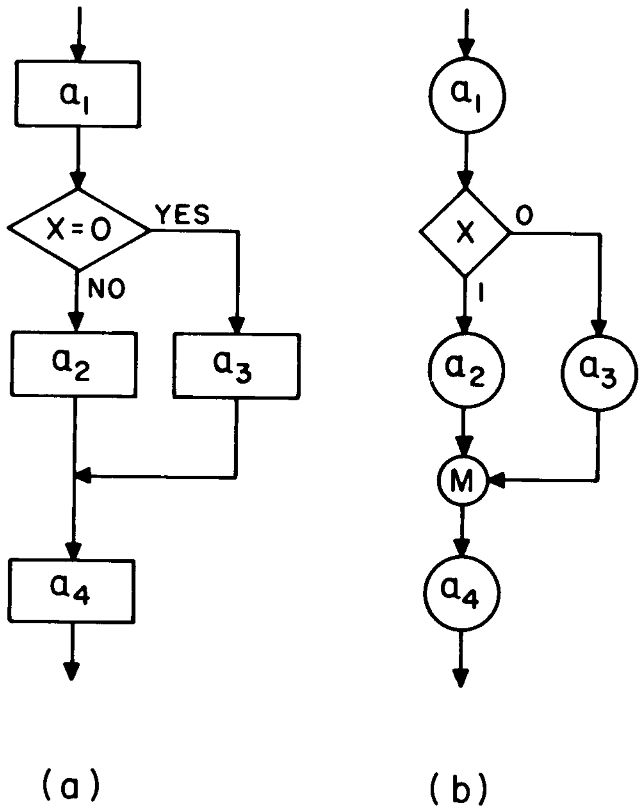


Figure 3 — Flow diagram (a) and control network (b) for a sequence containing a conditional branch

Figure 3 shows the flow diagram and control network for a sequence containing a conditional branch: the diagram defines the sequence a_1-a_3-a_4 when X=0 and the sequence a_1-a_2-a_4 when X=1. X is assumed to be a binary variable. The diamond-shaped element in Figure 3b is connected to the completion terminal for a_1 and the initiation terminals for a_2 and a_3. This element, called a decision (D) element, is a circuit that routes the completion signal coming from a_1 to the initiation terminal for a_2 or a_3 depending on the value of X.* The M element in Figure 3b is connected to the initiation terminal for a_4 and the completion terminals for a_2 and a_3. This element, called a merge (M) element, is a circuit that routes the completion signal coming from a_2 or a_3 to the initiation terminal for a_4.

Figure 4 shows the flow diagram and control network for a sequence in which operations a_2 and a_3 are to be executed in parallel. In the control network, the completion terminal for a_1 is connected to the initiation terminals for a_2 and a_3 so that the completion signal from a_1 will initiate both operations. The (R) element in the network is connected to the initiation terminal for a_4 and to the completion terminals for a_2 and a_3. This element, called a rendezvous (R) element, is a circuit that generates a control signal as soon as it has received a completion signal from both a_2 and a_3. Inclusion of the R element in the network guar-

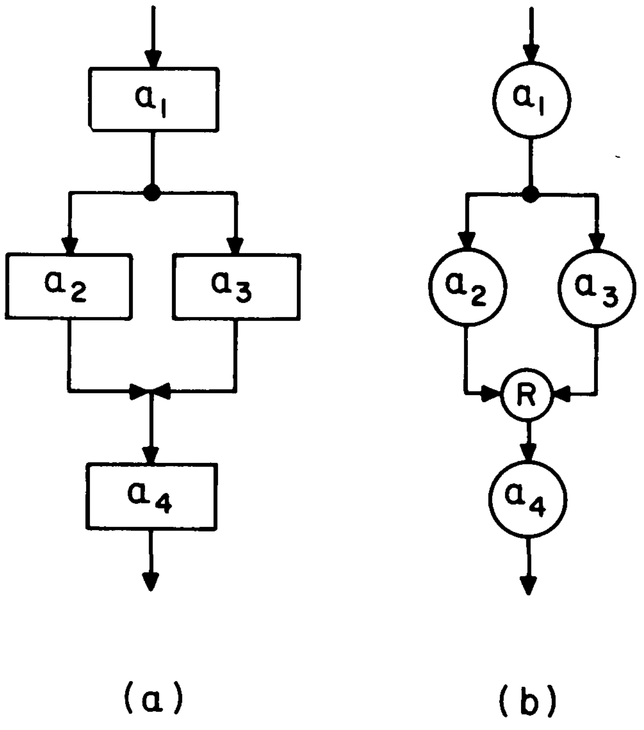


Figure 4 — Flow diagram (a) and control network (b) for a sequence involving parallel execution of operations

*Variable X is supplied to the D element at a terminal not shown in the figure antees that operations a_2 and a_3 will both be completed before operation a_4 is initiated.**

The correspondence between a flow diagram and a control network is not always one-to-one as implied by the preceding examples. The difference arises when an operation occurs in several sequences or in more than one place in the same sequence. In either case, independent access to the control terminals for the operation is required at several points in the network, and this cannot be accomplished by connections made directly to the control terminals. Consider, for example, independent sequences a_1-a_3-a_4 and a_2-a_3-a_5. Both require operation a_3, and direct connection to the control terminals of a_3 would result in the network shown in Fig. 5a. It is easy to see that this network does not describe two independent sequences. Proper implementation of the sequences, the network shown in Fig. 5b, uses a call (C) element to keep the two sequences separate. The C element has three pairs of control terminals, one of which connects to the control terminals of the operation to be multiply accessed. The other two pairs, indicated by the dotted lines, act as independent control terminals for the operation. When the completion signal from a_1 arrives at the C element, the element initiates operation a_3 and routes the completion signal from a_3 to the initiation terninal of a_4. When a completion signal from a_2 arrives, the C element initiates operation a_3 and routes the completion signal from a_3 to the initiation terminal of a_5.

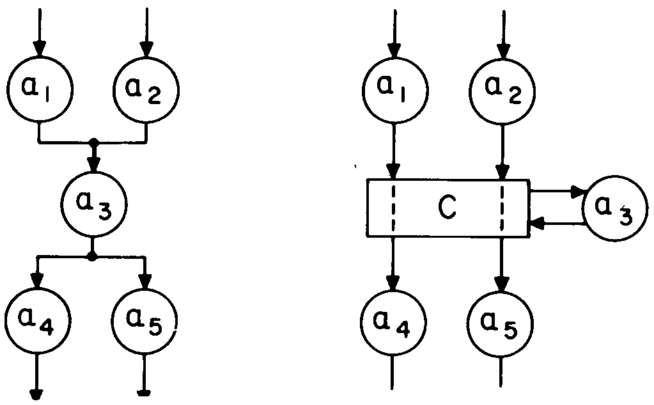


Figure 5—An improper way (a) and a proper way (b) of incorporating an operation's circuitry in two sections of control network

**It must be noted that if one of the operations is known to have a longer execution time than the other, its completion signal may be used to initiate a_4 and the R element can be removed from the network It is assumed in this paper that the execution time of an operation is unknown, and an R element will therefore always be used to terminate parallel processes

Data validation

Before any data processing operation may be executed, the latest value of the data to be used must be guaranteed present at the circuitry associated with the operation. This is a severe problem in macromodular systems since the physical separation between modules is not constrained and data propagation times are therefore unknown and unbounded. The problem is solved by a scheme called Data Validation (DV): when a data processing operation disturbs a source of data, the module does not generate a completion signal for the operation until the new data value has propagated to all parts of the system that may need it. Since subsequent data processing operations that use the data cannot occur until the completion signal is generated, the new data value is guaranteed available at the circuitry for these operations.

Data is carried from one module to another by means of special data cables or by inter-cell connections in the system frame. Included in each type of data path is an extra pair of lines carrying control signals associated with the DV process (Figure 6). Whenever a new data value begins to propagate along a data path, the data source transmits a control signal along line I of the path. This signal is generated slightly later than the propagating data value, and the data path and associated circuitry are designed so that the signal consistently lags the data. Hence, when the control signal reaches the other end of the data path, the circuitry there is guaranteed that the new data value has also arrived. As soon as this circuitry has assimilated the new data value, it sends a control signal back to the data source via line C of the path.

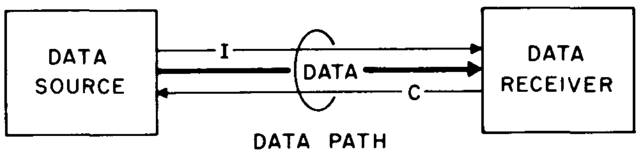


Figure 6—General structure of a data path

The I and C lines of a data path can be treated as control lines for an operation called dv. Figure 7a, for example, shows part of the internal control network of a module at the receiving end of a data path. The control lines labeled dv are the I and C lines of the data path, and operation a_1 is the operation (if any) performed by the module when notified of the arrival of a new data value. Figure 7b shows part of the internal control network of a module at the source end of a data path. It is assumed that operations a_1 and a_2 disturb the data source and the DV process is therefore performed after each of the operations. Figure 7c shows the same module designed for two data paths. The DV process for one of the paths is called dv_1 and the DV process for the other path, dv_2. As shown, dv_1 and dv_2 are executed in parallel.

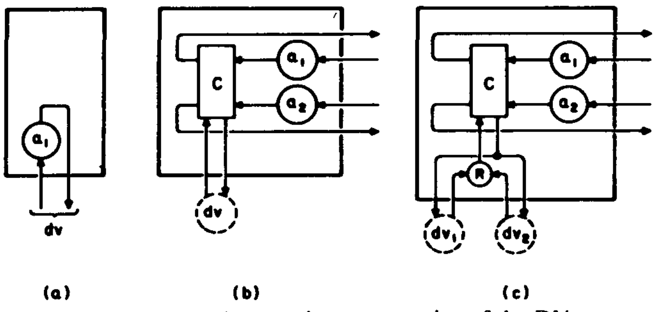


Figure 7 – Examples showing the incorporation of the DV process in the internal control networks of modules at either end of a data path

Wordlength extension

In order to simplify the control structure that a user of macromodules has to construct, system control has been made independent of data wordlength. Data processing modules are organized in parallel form with a wordlength modulus of 12 bits; a register module, for example, contains a 12 bit register and the adder module contains a 12 bit adder. If the user requires a wordlength of more than 12 bits, data processing modules can be cascaded to handle greater wordlengths by plugging them into laterally adjacent cells in the system frame. Inter-cell connections within the frame allow the internal control networks of the individual modules to link together to form a single control network for the entire array. This network, called the Wordlength Extension (WE) network, relegates operational control of the array to the control terminals on the rightmost module so that a data processing operation has but a single pair of control terminals regardless of the wordlength involved.

That portion of a WE network contained within a single module is called a WE segment, and the general form of the WE segment for a single data processing operation is shown in Figure 8. The terminals on the right are the control terminals associated with the execution of the operation in this module; the terminals on the left connect to the control terminals on the module in the left-adjacent cell that are associated with the execution of the operation in that module. The segment is designed according to the following rule: if it sends an initiation signal to the module in the left-adjacent cell, it must receive a completion signal from that module before it can generate a completion signal of its own. This rule guarantees that a segment will not generate a completion signal until all activated segments to its left have generated completion signals; hence, the completion signal generated by the rightmost segment is the completion signal for the entire array.

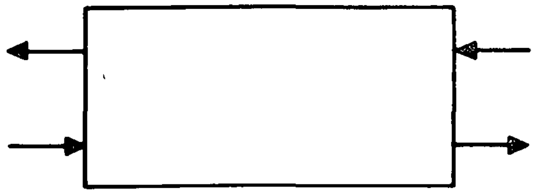


Figure 8 – Generalized representation of a WE segment

WE segments are composed of subsegments as shown in Figure 9. The segment labeled BD is a boundary delimiting network; its purpose is to inhibit communication with the left-adjacent cell if that cell does not belong to the array. The flow dia-

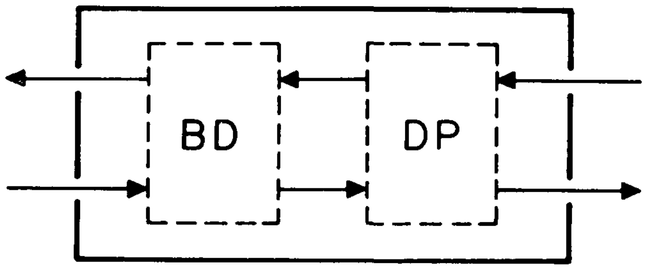


Figure 9 – A WE segment designed in terms of subsegments.

gram for the BD segment is shown in Figure 10 and is the same for all WE segments. The segment labeled DP is that portion of the WE segment which carries out the data processing operation in the module.

Fig. 10 and is the same for all WE segments. The segment labeled DP is that portion of the WE segment which carries out the data processing operation in the module.

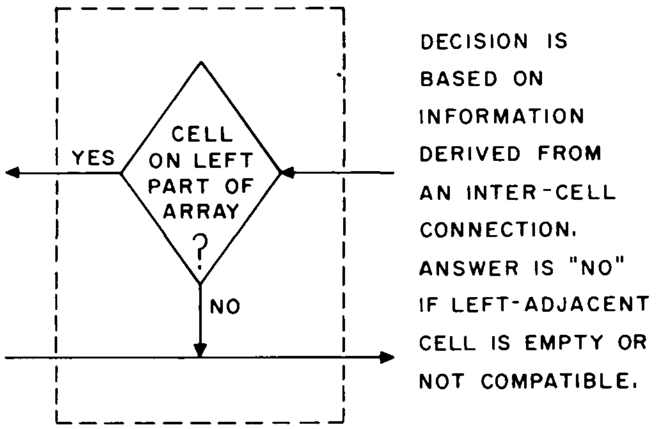

YES
CELL ON LEFT PART OF ARRAY ?
NO
DECISION IS BASED ON INFORMATION DERIVED FROM AN INTER-CELL CONNECTION. ANSWER IS "NO" IF LEFT-ADJACENT CELL IS EMPTY OR NOT COMPATIBLE.
Figure 10 – Flow diagram of the BD subsegment

Figure 11a, for example, shows the DP network for the operation CLR (set all bits to zero). The network here is a parallel one, a control signal being sent to the left at the same time that the CLR operation for the module is initiated. The R element guarantees that the network does not generate a completion signal until it has received a signal from the left. Figure 11b shows a serial network for the same

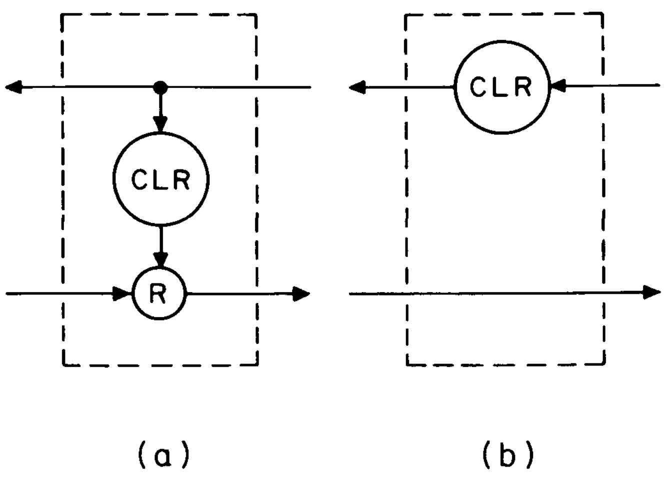


Figure 11 – A parallel network (a) and a serial network (b) for the DP subsegment associated with the operation CLR

operation. In this network, a signal is sent to the left only after the CLR operation for the module is completed; hence the network can generate a completion signal as soon as it receives a signal from the left. Figure 12 shows the flow diagram of a DP

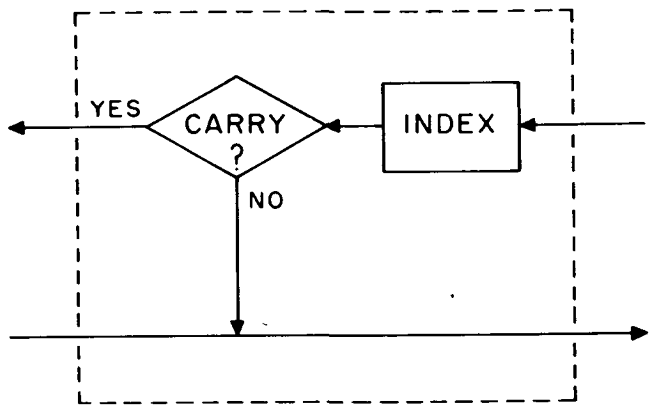


Figure 12 – Flow diagram of a DP subsegment for the operation INDEX

segment for the operation INDEX (add 1 to the value of the data). In this segment, a signal is sent to the left only if the INDEX operation in this module indicates a carry into the next module.*

*The most significant bit is in the leftmost module of the array.

Transfer of data

The transfer of data into a register is accomplished by means of a module called a data gate Figure 13). The terminals on this module are the control terminals for the transfer operation, and when the operation is initiated, the data on the lateral data path is transferred into the register. Data gates may be cascaded as shown in Figure 14 so as to allow data from any number of sources to be transferred into a register. The data paths running between the data gate modules allow each of these modules to communicate with the register.

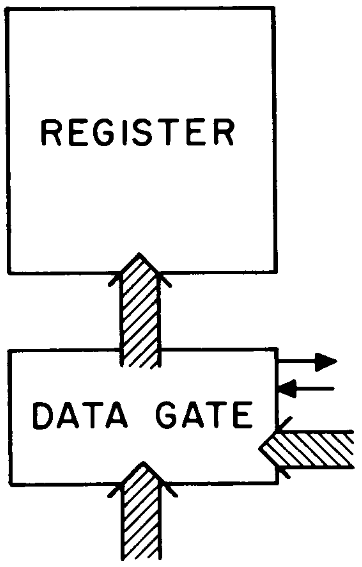


Figure 13 – Proper interconnection of a data gate module and a register module Broad arrows represent data paths

The logic associated with the data path input of the register is shown in Figure 15. As indicated, the dv operation for the path causes the newly arrived data value to be loaded into the register. The general transfer process therefore consists of two steps: (1) the data to be transferred must be gated onto the data path input of the register, and (2) the dv operation for the path must then be initiated. These functions are provided by the data gate module at the other end of the data path. When a signal arrives at the data gate's initiation terminal (Figure 16), a flip-flop G is set ($1 \rightarrow G$) which gates the data of the lateral input path onto the output path, and the dv operation is then initiated. At the conclusion of the dv operation, the flip-flop is reset ($0 \rightarrow G$) so that the lateral path is no longer connected to the output path, and the module then generates a completion signal. When the transfer process is not being executed by the module, the output path is connected to the bottom input path shown in Figure 16 so that modules located

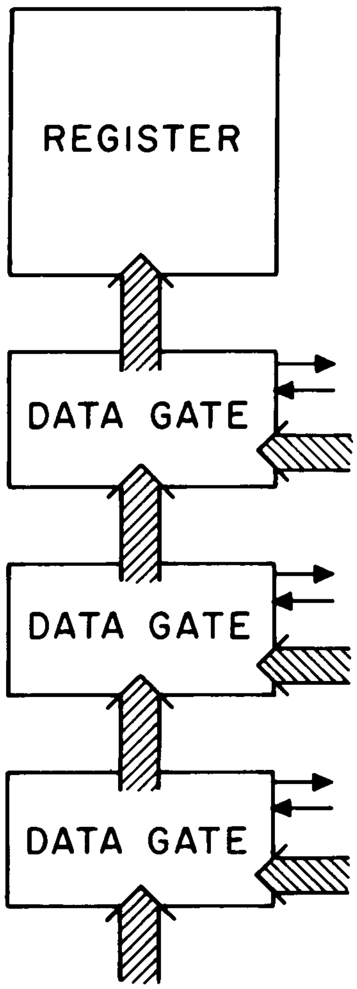


Figure 14 – Data gate modules cascaded to allow data from three different sources to be transferred into a register module

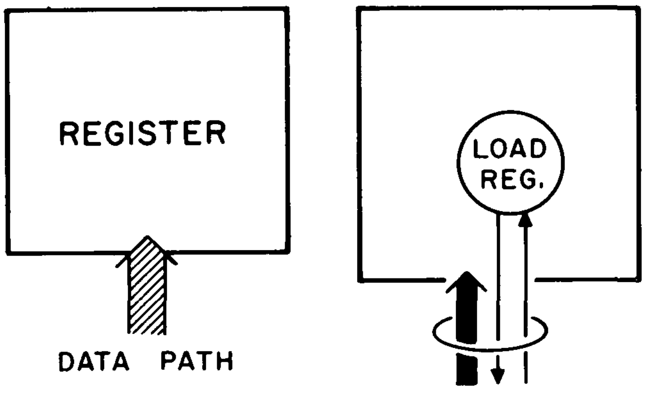


Figure 15 – Register logic associated with the transfer process

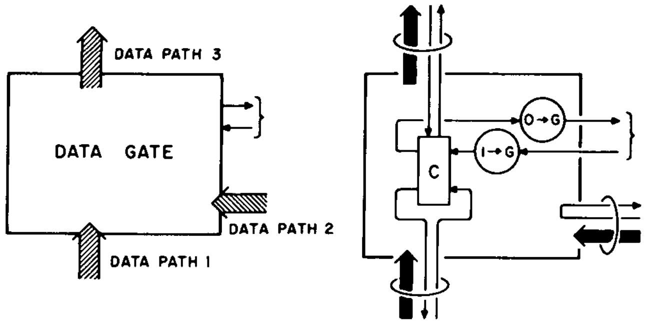


Figure 16 – Data gate logic associated with the transfer process

below this module may communicate with the register. The dv process associated with the bottom input path initiates the dv process for the output path, thereby insuring that the dv process initiated by a lower module will propagate through this module and on to the register. The C element in the figure keeps the transfer sequence initiated by this module separate from transfer sequences initiated by lower modules.

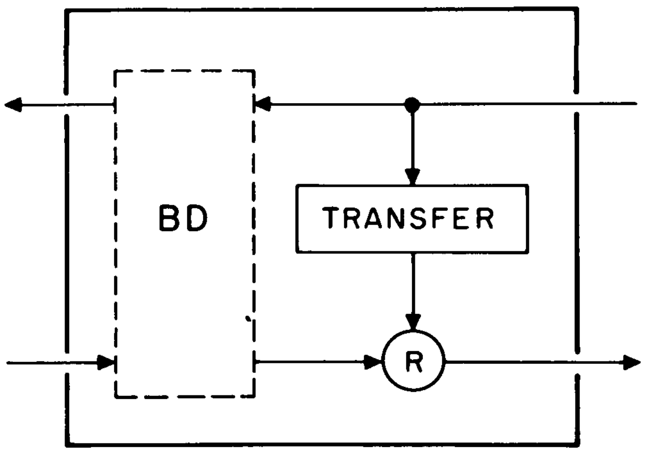


Figure 17 – Flow diagram of the WE segment of a data gate module

Wordlengths of more than 12 bits are handled by laterally cascading data gate modules as described in the section on Wordlength Extension. The WE segment for a data gate is shown in Figure 17, and Figure 18 shows the assemblage of modules required for 36 bit transfers from three different sources.

Implementation

In present macromodular design, an initiation terminal accepts a binary input, a completion terminal produces a binary output, and a control signal is a change in value, either from 1 to 0 or from 0 to 1. The control elements are realized in asynchronous fundamental mode level logic form, and the state and output tables for each are given in Figure 19.

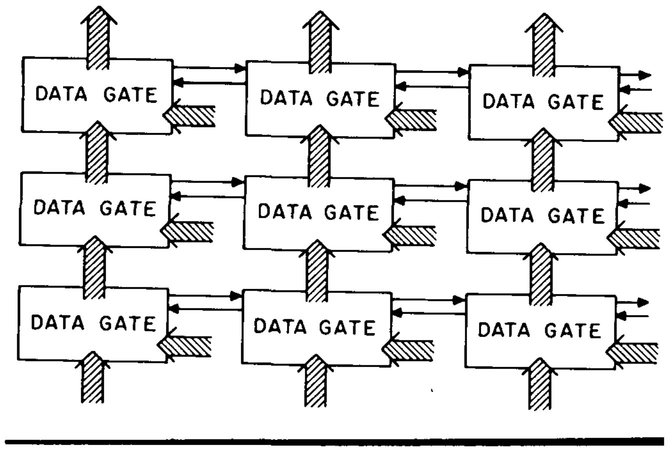


Figure 18—The assemblage of data gate modules required for transferring 36 bit words from three different data sources

CKNOWLEDGMENT

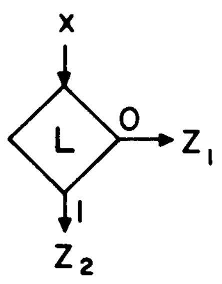
The authors wish to express their gratitude to A.Anne and Y.H. Chuang for their help in the development of the ideas presented here.

REFERENCES

1 W A CLARK
Macromodular computer systems Proc SJCC 1967

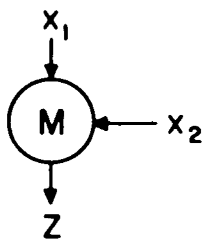
2 S M ORNSTEIN M J STUCKI W A CLARK
A functional description of macromodules
Proc SJCC 1967

(a) DECISION ELEMENT



	Lx 00	01	11	10	$Z_1 Z_2$
A	Ⓐ	D	B	Ⓐ	0 0
B	C	Ⓑ	Ⓑ	A	0 1
C	Ⓒ	B	D	Ⓒ	1 1
D	A	Ⓓ	Ⓓ	C	1 0

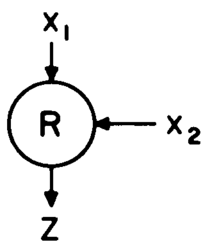
(b) MERGE ELEMENT



x_1 \ x_2	0	1
0	0	1
1	1	0

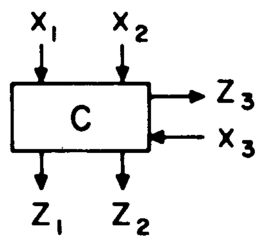
Z

(c) RENDEZVOUS ELEMENT



	$x_1 x_2$ 00	01	11	10	Z
A	Ⓐ	Ⓐ	B	Ⓐ	0
B	A	Ⓑ	Ⓑ	Ⓑ	1

(d) CALL ELEMENT



x_1 \ x_2	0	1
0	0	1
1	1	0

Z_3

	$x_3=0$ x_1x_2 00	01	11	10	$x_3=1$ 00	01	11	10	$Z_1 Z_2$
A	Ⓐ	Ⓐ	—	Ⓐ	Ⓐ	B	—	D	0 0
B	A	Ⓑ	C	—	Ⓑ	Ⓑ	Ⓑ	—	0 1
C	—	Ⓒ	Ⓒ	Ⓒ	—	B	Ⓒ	D	1 1
D	A	—	C	Ⓓ	Ⓓ	—	Ⓓ	Ⓓ	1 0

Figure 19 – State and output tables for the control elements

MACROMODULAR SYSTEM DESIGN

Wesley A. Clark and Charles E Molnar

TECHNICAL REPORT NO. 23

April, 1973

Computer Systems Laboratory
Washington University
St. Louis, Missouri

This work has been supported by the Advanced Research Projects Agency of the Department of Defense under Contract SD-302 and by the Division of Research Resources of the National Institutes of Health under Grant RR-00396.

ABSTRACT

Macromodules are "building blocks such as registers, adders, memories, control devices, etc., from which it is possible for the electronically naive to construct arbitrarily large and complex computers that work." In the seven years since this statement was made in the presentation of a program then being undertaken at Washington University, the design and construction of a several hundred unit macromodular inventory has been accomplished, and some 60 or 70 widely different "computers that work" have been constructed, used and dismantled. This report briefly reviews the operational aspects of this inventory and our experience in working with it, and illustrates some of our present attitudes and values concerning the macromodular approach, with special reference to biomedical research applications.

TABLE OF CONTENTS

INTRODUCTION

Macromodules are "building blocks such as registers, adders, memories, control devices, etc., from which it is possible for the electronically-naive to construct arbitrarily large and complex computers that work" (Ornstein, et al, 1967, p. 337). In the seven years since this simple statement was made in the presentation of a program then being undertaken at Washington University, the design and construction of a several hundred macromodular unit inventory have been accomplished Over the last three years, some 60 or 70 widely different "computers that work" have been constructed, used, and dismantled.

In this report we will review briefly the operational aspects of this inventory and our experience in working with it, and illustrate some of our present attitudes and values concerning the macromodular approach to the implementation of algorithmic systems, with special reference to biomedical research applications.

Why should one want to construct his own computer? Is it not clearly better to write a program for a general-purpose machine which will solve the problem at hand? The answers, of course, depend on what is meant by "computer" and on how well the program fits the available stored-program machine. Problem solution often requires only a relatively well-understood computation or processing task which can conveniently be programmed and run on a general-purpose computer. There are many problems, however, which do not have convenient computational solutions in the ordinary sense; they may require processing of data at expensively high speeds, for example, or give rise to unusually demanding algorithms which result in grossly inefficient operation on the so-called general-purpose computer. Some research problems involving large and expensive computational tasks exhibit a particularly inconvenient combination of traits, requiring dedicated machinery for good progress yet at the same time not being well understood in either form or value. The research worker, if he opts to undertake the task of solution at all, runs the considerable risk of premature or excessive commitment to what might well turn out to be the wrong way of doing things, or perhaps, the right way of solving the wrong problem.

The macromodular approach is an attempt to introduce a new dimension in problem formulation and solution. It permits the research worker to think in terms of computational, data processing, or algorithmic machinery which efficiently fits the problem as it is initially understood, can be adjusted or rearranged as the work progresses, and finally, can be dismantled for subsequent use elsewhere and either conveniently forgotten or gratefully memorialized as circumstances warrant. Naturally, this approach makes the most sense when the computations are large enough and will run long enough to justify the setup time involved. It should not be assumed, however, that macromodular systems are necessarily full computers in the ordinary sense. We have often found that combining macromodular equipment with other parts of a larger system including a small stored-program computer is most efficient and convenient, with the macromodular subsystem handling those parts of the total processing task in which the payoff of specialization and flexibility is relatively great. The reader may wish to refer to the program presentation papers (Ornstein, et al, 1967), (Stucki, et al, 1967), for a more complete exposition of the conceptual framework, which has remained essentially intact throughout the course of development despite inevitable adjustment in detail. Much of the evolution of the development is recorded, albeit in an uneven and incomplete form, in a series of Computer Systems Laboratory Technical Memoranda and Technical Reports (Computer Systems Laboratory, 1972).

It is the dedication of the staff of the Computer Systems Laboratory over a period of several years which has given substance to the macromodular system concept. The Computer Systems Laboratory was formed in 1967 to continue the system development work of the Computer Research Laboratory, which had been established at Washington University in 1964 under the direction of William N Papian The principal approaches to macromodular system architecture and logic were developed early in the program by Mishell J. Stucki and Severo M. Ornstein. The detailed logic design of most of the macromodules described here was done by or under the supervision of Mr. Stucki, with contributions from Antharvedi Anné, Henry Y.H. Chuang, Kazuaki Harada, Maurice L. Pepper, Jr., Archie D Richardson, and Mary Allen Wilkes.

Messrs. Asher S. Blum, Thomas J. Chaney, George R. Couranz, Howard C. Lewis, Warren M. Littlefield, Richard E Olson, and Oliver F. Stone shaped the early stages of engineering development, which led to a later phase of manufacturing and production design in which the contributions of Robert J. Arnzen, Gerald C. Johns, George C. Meyer, Fred U. Rosenberger and David L. Stewart were added. Important stimulation and criticism were generated through system design studies by R Martin Arthur, Nadine Bicket, Jerome R. Cox, Jr., Richard A. Dammkoehler, Robert A. Ellis, Mark A. Franklin, Mars J. Gralia, Thomas H. Jacobi, Garland R Marshall, Eitan Sadeh, Bruce F. Spenner, and Donald F. Wann. Patricia B. Savage, Christine E. Dickson and John A. Newell played a vital role in developing critical documentation.

The energetic and enthusiastic technical support of Norman T. Kinch and his staff in providing all the vital services needed to carry out a major program of design, development and fabrication has played an essential role, as has the unfailing administrative support of Edward L. MacCordy, Dedie Fries, and the central administration of Washington University. The patience and understanding of our sponsors are gratefully acknowledged.

CHARACTERISTICS OF MACROMODULAR SYSTEMS

Assembly Procedure

How does one use macromodules to build a computer, given a statement of the problem? After identifying the relevant computing tasks, the designer sketches a suitable structure in block diagram form based on the logic of macromodules, grouping units into various storage and data processing substructures which seem to be useful, and specifies data transfer pathways among these groups. He then draws a flow diagram specifying the sequences and concurrences of the basic operations to be carried out within the resulting structure. When the design appears to be approximately correct, though not necessarily complete, he is ready to build a macromodular system. Sometimes it is more convenient to design the system, or pieces of it, in the frame itself with the actual parts, documenting it after the fact; and sometimes it is easier to re-invent the system than it is to document it at all.

We have come to identify as "macromodular" not just the operational electronics packages, but rather all of the physically distinct elements of the inventory which must be assembled into a working macromodular system. This viewpoint and the kinds of elements dealt with are best understood by outlining the general assembly procedure

First, a frame of suitable size is formed by installing a *Frame Block* on top of a *Pedestal Unit*, stacking additional Frame Blocks on top of the first until the estimated working size is achieved (Figure 1). This is the only operation requiring a tool (a screwdriver). The resulting structure provides a regular array of mounting positions or cells into which other macromodular elements will be inserted, each Frame Block contributing 16 cells (4 x 4). Wider frame structures are build by leveling a second Pedestal in a position contiguous with the first, stacking a second column of Frame Blocks on this Pedestal, and interconnecting corresponding cell tiers (through connectors on the ends of each lateral structural member) by means of bridging *Coupling Units* There is no intrinsic limit to the width of a frame.

This part of macromodular system construction is relatively heavy work of no great fun.

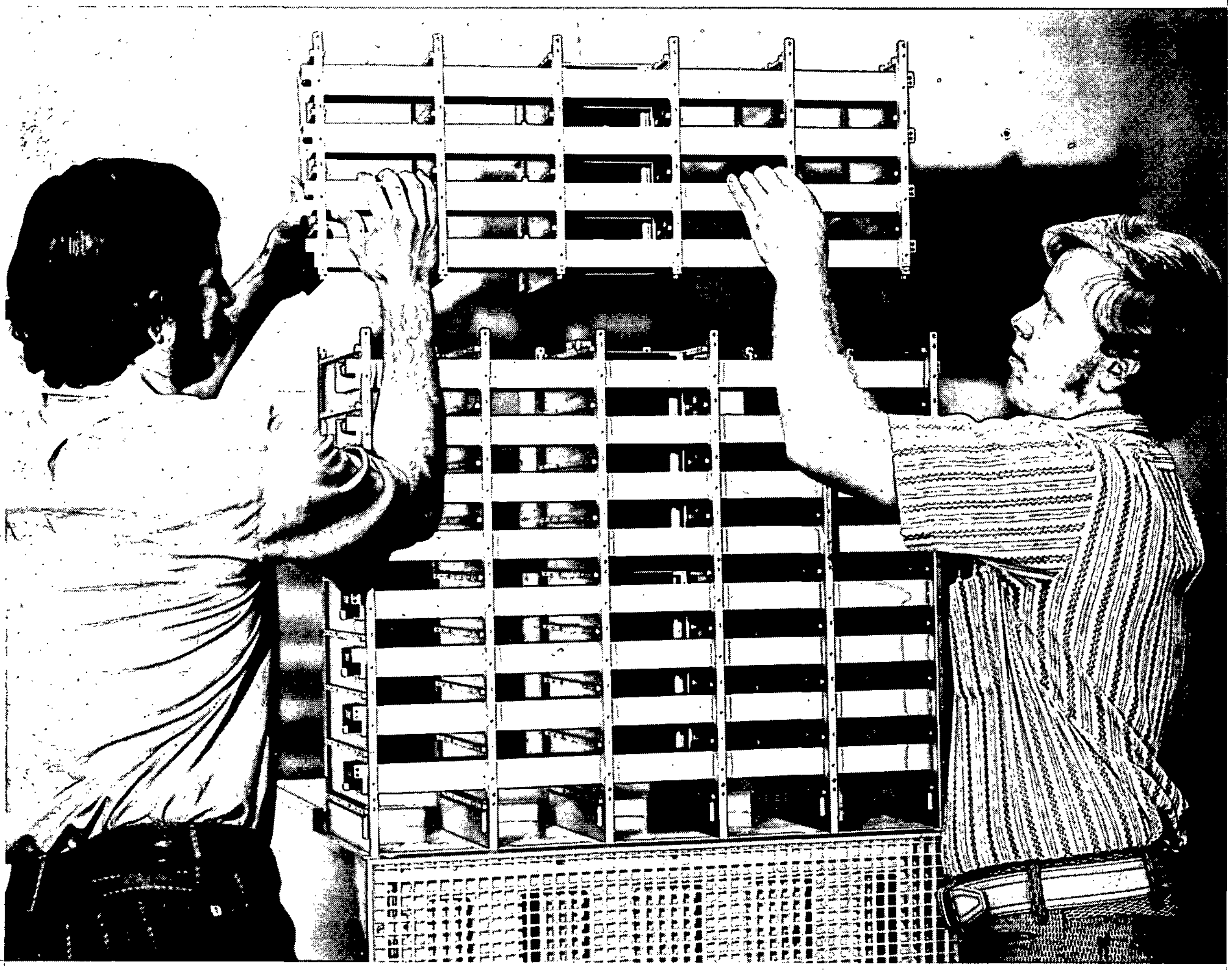


Figure 1

A third Frame Block being added to a frame. Each Frame Block carries within its built-in fan subassembly and structural members all necessary cooling air ducts, signal paths for lateral (horizontal) communication between adjacent cells, and power and system-wide signal paths which lead from the Block below (ultimately from the Pedestal) and are distributed in turn to all cells and to the Block above. The Pedestal Unit at the bottom houses power supplies, distributes system-wide control signals, and provides mechanical stability. Up to eight Frame Blocks can be stacked on a Pedestal.

Next the system designer/assembler selects appropriate cable-to-connector adapter units called *Faceplate Boxes*, attaches to the face of each a color-coded *Overlay* (Figure 2), bearing a friendly and useful label such as LOAD, CALL, R = D?, etc., and snaps the boxes into the frame from the front (Figure 3) to form patches of filled cells in which adjacency and ordering of the labelled functions begin to define the logic of the system under construction. "Data processing" functions such as ADD, COMPARE, REGISTER, MEMORY, etc., are implemented in 12-bit segments; data processing patches are called *Manifolds*, structures in which contiguity of the boxes assures appropriate "automatic" interconnection via implicit vertical and lateral pathways. Systems typically require several different manifolds. Manifolds for the processing of numbers greater than 12 bits in "width" are assembled by extension to the left, filling cell columns adjacent to the column defining the manifold with appropriate Faceplace Boxes bearing "extender" overlays for each function, thus forming a manifold of 24 bits, 36 bits, etc. It is harder to make a 119-bit manifold than one of 120 bits, but there is no intrinsic upper limit to manifold width.

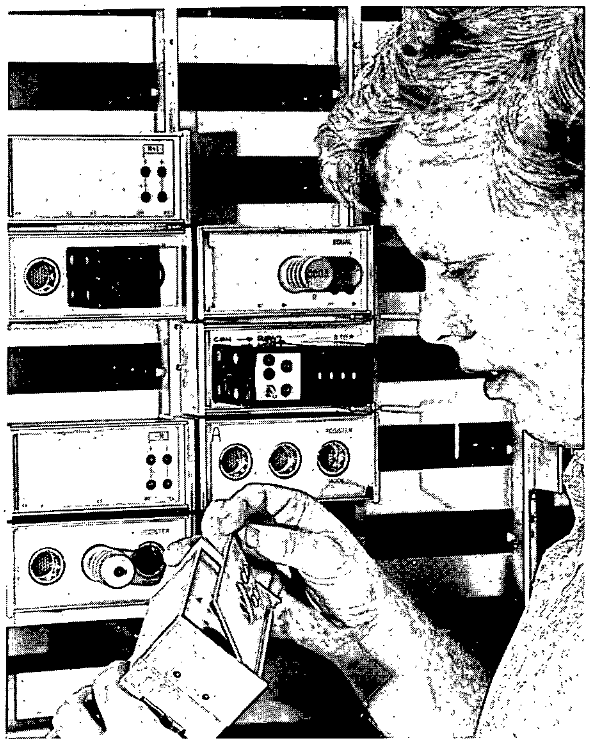


Figure 2

An Overlay being mounted on a Faceplate Box. A function code carried by the Overlay plate in the form of small punched holes along the bottom edge is sensed by switches in the Faceplate Box and transmitted to the corresponding Electronics Package. Indicator and Parameter Blocks and a Miniconsole can be seen plugged into Data Ports in the background.

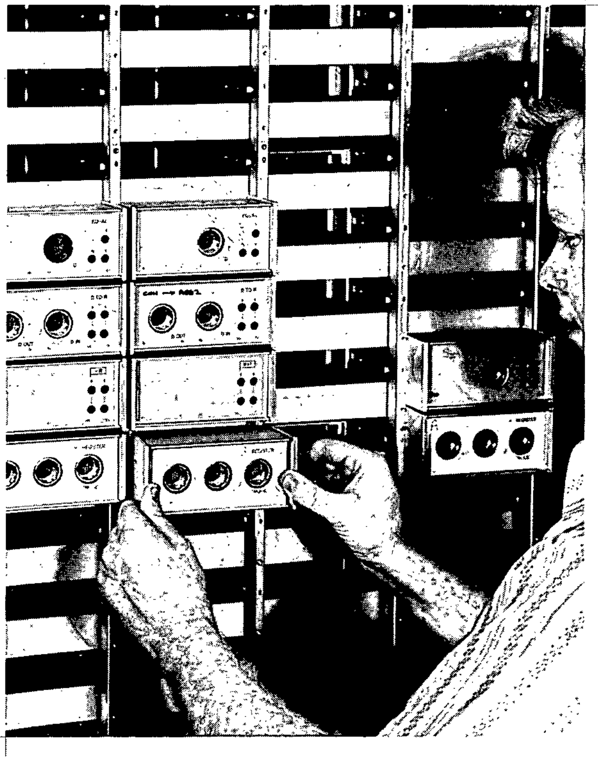


Figure 3

The Faceplate Box being inserted into a Frame-cell. Faceplate Boxes provide a cable-to-connector adapter and establish vertical interconnections between adjacent Electronics Packages.

Faceplace Boxes for anticipated control functions such as CALL, MERGE, RENDEZVOUS, etc., are similarly mounted in the frame, sometimes with rearrangement of previously installed boxes for reasons of convenience, esthetics, oversight, or caprice. Adjacency is generally irrelevant in the placement of control function Faceplate Boxes.

The designer has now assembled in the vertical plane a giant, colorful plugboard whose patchy facade presents many cable connector sockets, which appear through holes in the overlays. He then makes any required data pathway interconnections not already provided by the implicit lateral or vertical busses (e.g., those from manifold to manifold) by means of *Data Cables* of convenient lengths. At this point some useful monitoring or control modules of yet another kind may be plugged in here and there in place of cables: *Indicator Blocks* holding 12 lights may be plugged into any output; *Parameter Blocks* holding four octally-coded thumbwheel switches, into any input.

Next, following the flow diagram exactly, the control network is "wired up" by plugging in *Control Cables* from the control output of one module to the control input of the module whose function is next to be evoked in sequence. *Miniconsoles* can be "spliced" into any control signal pathway to interrupt and indicate an incident signal (a completion signal from the previously evoked function) and to provide for manual initiation of the continuation control signal.

With the final cabling steps of leading data and control cables to an *Interface Unit* (through which the system can be connected to a separate conventional stored program machine for further support if required), the chaining together of the Pedestals for power sequence control from a central *Power Control Console*, and plugging the required power cables from the nearest AC power outlets into each Pedestal, the task of wiring the macromodular system is finished (Figure 4). To complete the entire structure, it remains only to insert the required *Electronics Packages* into the appropriate cells from the rear of the frame (Figure 5), and to install the required number of power supply slugs (Figure 6).

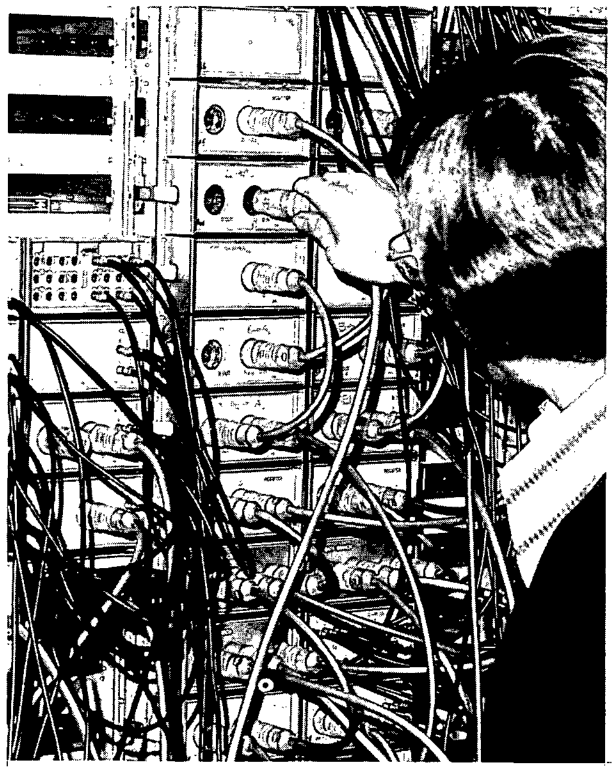


Figure 4

Data Cables (gray) carry 12-bit data and data validation signals from point to point with great flexibility; the placement of Control Cables (black) determines the sequence in which operations take place.

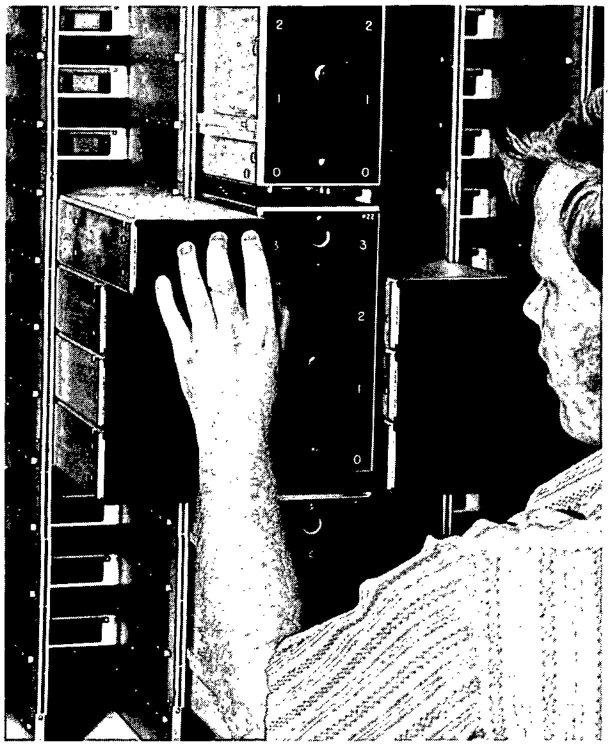


Figure 5

Electronics Packages, typically containing 80 integrated circuits of the MECL II family, are inserted into the frame from the rear after Faceplate Boxes and cables have been installed.

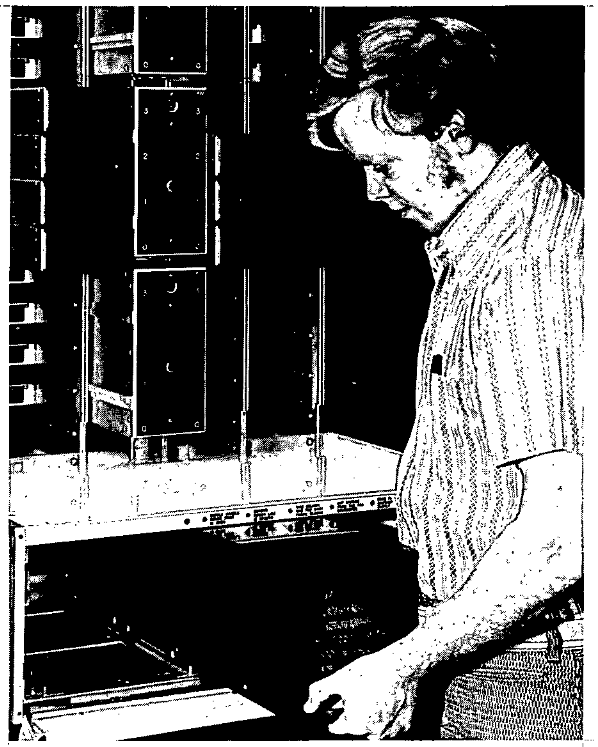


Figure 6

Indicator lights on the rear of the Pedestal Unit signal a need for additional power supply units. Up to three units can be plugged into the pedestal, providing a maximum power level of 2000 watts per frame.

There are seventeen different Electronics Package types, each subserving a class of functions and color-coded to match the corresponding Overlay on the Faceplate Box to which it is to be connected on insertion into the frame. Thus, for example, the LOGIC Electronics Package implements the 16 Boolean functions of two variables. When plugged from the rear into a cell holding an EXCLUSIVE OR-labelled Faceplate Box, the Exclusive Or function will be the only one evoked by an incident control signal. Manifold Electronics Packages carry a segment of the vertical bus pathways as well, together with the associated amplifiers, so that the act of stacking functions within the manifold builds the required bus automatically.

The designer of, say, a 50-cell macromodular system has now, in the course of an hour or so, put together a fairly large amount of equipment (which, incidentally, will have required the pluggable-contact interconnection of about 20,000 electrical pathways). The power is then turned on and operation is begun.

The promise of the macromodular approach is that no logically-proper system assembled in this way will fail to operate algorithmically. If the algorithmic behavior of the system is not what the designer intended, then it can be presumed that the fault is attributable to the logic, not to a misbehavior of the electronics or mechanics or of any other aspect of the hardware or of the signals represented therein. It is not necessary to assure the incredulous engineer among the readers of our awareness that this promise can never be completely fulfilled. It should be pointed out, however, that we have come workably close--an achievement which has required the carefully coordinated solution of a great many design and engineering problems.

Macromodular Cell Logic

With the problems of hardware, in effect, presolved, the designer of a macromodular system concerns himself only with systems logic, conceptually and physically manipulating elements of macromodular hardware much as he would symbolically manipulate the primitives of stored-program code at the assembly language level, in both cases taking the *algorithmicity* of the hardware for granted. We speak of the macromodular primitives in terms of the logic of the cells, informally and rather inexactly thinking of the diverse hardware elements, combined to make a given cell work, as "the macromodule" The principal characteristics of the kinds of cells that can be composed from the elements of the present inventory are summarized below:

1. REGISTER - Storage of one 12-bit number and a flag bit; 2 data cable outputs; vertical data bus communication with cells above.
2. LOAD - Parallel transfer of number on data cable input via down bus; number reproduced on data output cable to facilitate chaining.
3. LOGIC - Any of the 16 Boolean functions of 2 variables, one variable supplied by a data cable input and the other from below (on the up bus), bit-columns affected are determined by a second data cable input; result transferred via down bus.
4. ARITHMETIC - The Add class functions (add, subtract, count, etc.) with or without overflow (captured in the flag bit of the underlying register); one variable supplied by a data cable input and the other from below (on the up bus); result transferred via down bus.
5. SHIFT - Unit, bidirectional scaling and rotating functions with variations in end-bit treatment; result transferred via down bus.
6. COMPARE - Arithmetic comparison functions; zero value, positivity, flag status detection, etc.; one variable supplied by a data cable input and the other from below (on the up bus); bit matching in bit-columns determined by a second data cable input.
7. DECODE - One-to-eight control path decoding of 3 bit-columns of the up bus selected by a data cable input.
8. MULTIPLY - Formation of double-length product of the number on the up bus with the number on the data cable input; selectable half of the result transferred via down bus, the other half via data cable output. Requires double cell.
9. DATA BRANCH - Provides 2 data cable outputs which replicate the number on a data cable input; presents the same number to up bus.
10. D/A UNIT - 2 storage elements, each element capturing a 12-bit number appearing on the up bus upon external control signal command. A third control signal input causes conversion of these two numbers to voltages which appear on two output coax cable connectors, while a fourth control signal input produces an "analog-signals-ready" pulse on a third output coax connector.
11. MEMORY UNIT - 4096-word, 1-μsec core memory. Address and data in and out are passed by vertical busses. Requires double cell.
12. UNIT MEMORY CONTROL - Surmounts one MEMORY UNIT; provides for address input and memory output on data cables with control ports for "Read"; presents register-like vertical bus interface.
13. GENERAL MEMORY CONTROL - Surmounts a stack of MEMORY UNITS, providing 2 data cable inputs for address, together with data input and data output cables; contains a resident high-speed scratchpad memory of 256 words; operating mode selected by FUNCTION CALL unit; requires double cell

14. CALL UNIT - 2 independent Call sections, each section providing 3 sets of control ports (in each set, one input and two outputs) together with a fourth common set. For each section a control signal on one of the input (Call) ports produces a signal on the common output port. A subsequent completion signal on one of the two common input ports produces an output signal on the corresponding output port of the Called set.
15. FUNCTION CALL UNIT - Combines Call with parameter plug specification of the code number of the function to be performed by the data processing macromodule to which it is connected by special cable.
16. MERGE/RENDEZVOUS - Sets of six control elements, each element combining asynchronous input control signals on 2 control cable ports in either MERGE ("OR") or RENDEZVOUS ("AND") relationship and providing 2 output control cable ports on both of which an output signal appears when the given condition is met.
17. INTERLOCK - Sets of four interconnected elements, designed to control access to a common storage unit or processing structure used by several concurrently running macromodular processors.

System Design Principles

We now proceed to illustrate the conceptual manipulation of macromodules as cellular block diagram elements in the following sequence of figures. Each figure represents a logical structure which is somewhat more complex than the one of the preceding figure within the sequence. The functions chosen are not remarkably interesting ones, nor is their implementation here always optimum. Furthermore, we have not attempted to illustrate exhaustively the use of each type of macromodule. Our intention here, rather, is to give the reader some feeling for the dynamics of rearrangement and growth, aspects which might ideally be presented in motion picture form.

We begin with the basic storage element, the REGISTER, and the first figure (Figure 7) shows what one might find after installing the Register Faceplate Box, Overlay, and Register Electronics Package in a handy frame cell position, and then turning on the power and plugging Indicator Blocks into both Data Cable output connectors. As indicated, the two data outputs are identical, showing, in this case, the number 5302 (octal; in decreasing numerical order, the 12 bits are read left to right, top to bottom. Dark circles indicate ones.) This number is of no significance, and merely represents the flip-flop state biases of the particular Register Electronics Package chosen.

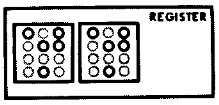


Figure 7

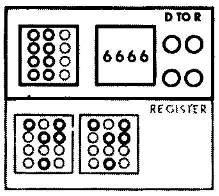


Figure 8

In Figure 8, a LOAD unit, particularized to the data transfer function (D TO R), has been installed in the cell above the REGISTER, the power has been turned on, and a Parameter Block set to 6666 has been plugged into the Data Input connector; another Indicator Block plugged into the duplicate Data Output connector shows the same value. The initial bias number reappears at the REGISTER as before

In Figure 9, a third cell has been filled with a suitable unit (another LOAD, for example) to provide a convenient, powered Data Connector into which a Miniconsole has been plugged. Control Cables are connected from the Control Output of the Miniconsole to the Control Input (the initiation terminal) of the D TO R cell, and from the Control Output (the completion terminal) of the D TO R cell back to the Control Input of the Miniconsole. The figure shows the outcome of flipping the switch on the Miniconsole, an action which sends a control signal to the D TO R unit, which causes the number set in the Parameter Block to be transferred to the REGISTER via the vertical data bus interconnecting the two cells, after which a completion signal returns to the Miniconsole, whose indicator lights then show that the operation has been carried out Changing the setting of the thumbwheel switches of the Parameter Block has no further effect on the REGISTER until the Miniconsole switch is flipped again. (The duplicate data Indicator Block does, of course, follow the setting of the Parameter Block exactly regardless of the action of the Miniconsole.)

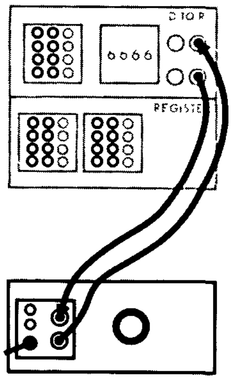


Figure 9

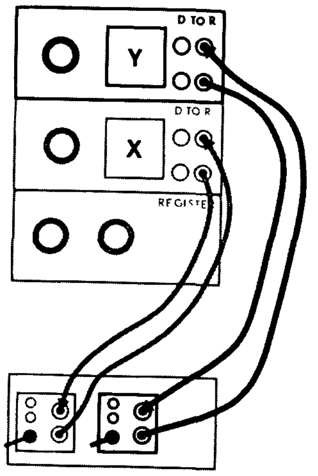
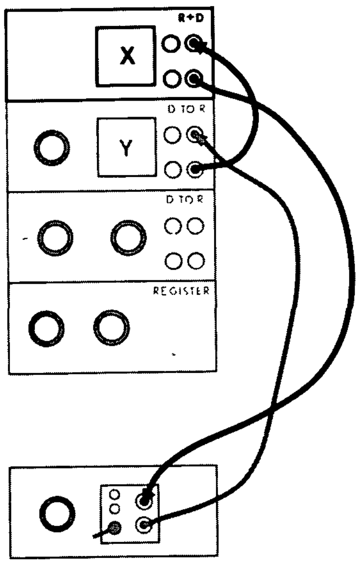
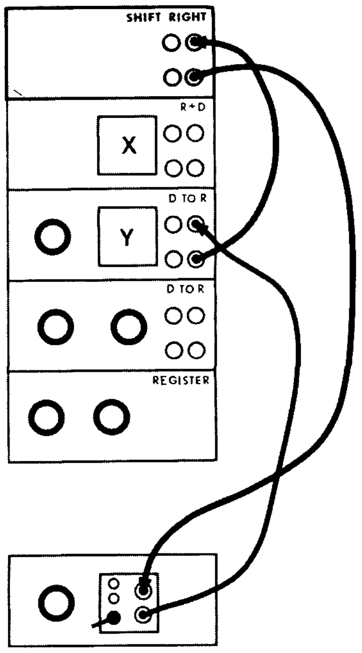


Figure 10

In Figure 10, a second D TO R unit with another Parameter Block has been stacked above the first and cabled to a second Miniconsole. The Indicator Blocks have been removed The parameter X can be transferred to the REGISTER by flipping the left Miniconsole switch, the parameter Y, by flipping the right Miniconsole switch. These actions are independent of one another so long as they are not asked to occur simultaneously. The units of the REGISTER manifold, now three cells high, are interconnected by the implicit vertical bus pathways.

An ARITHMETIC unit particularized by an appropriate Overlay to the simple addition function R + D has been added to the manifold in Figure 11, with the Parameter Block holding parameter X moved to its Data Input connector. Control cables have been rearranged so that flipping the Miniconsole switch now causes a *sequence* of actions to occur. first, the parameter Y is loaded into the REGISTER, and second, the parameter X is added to the contents of the REGISTER, leaving the sum X + Y in the REGISTER and sending a completion signal back to the Miniconsole. The lower D TO R unit serves only to relay data and control signals via the vertical bus, but is otherwise unused.





In Figure 12, a SHIFT RIGHT unit has been installed and the control cabling rearranged to show how the result $\frac{Y}{2}$ can be formed in the REGISTER

By substituting a SHIFT LEFT unit for the unused D TO R unit and recabling the control as in Figure 13, the result $\frac{1}{2}(4Y + X)$ can be formed in the REGISTER after the following sequence of steps is initiated by flipping the Miniconsole switch:

1) Y to R (Y to REGISTER),
2) 2R to R (SHIFT LEFT),
3) 2R to R (SHIFT LEFT again),
4) X + R to R,
5) $\frac{1}{2}$R to R (SHIFT RIGHT).

The reader will perceive that further rearrangements of this kind will result in the generation of other simple functions.

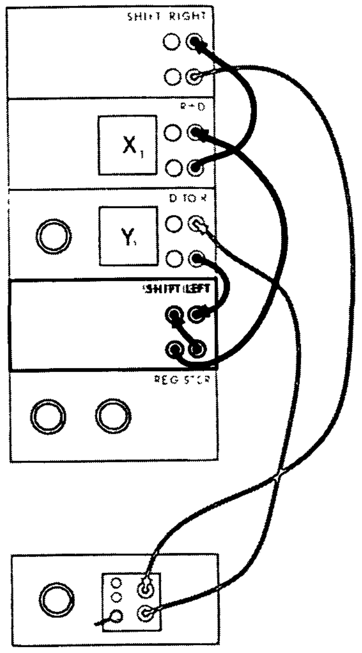


Figure 13

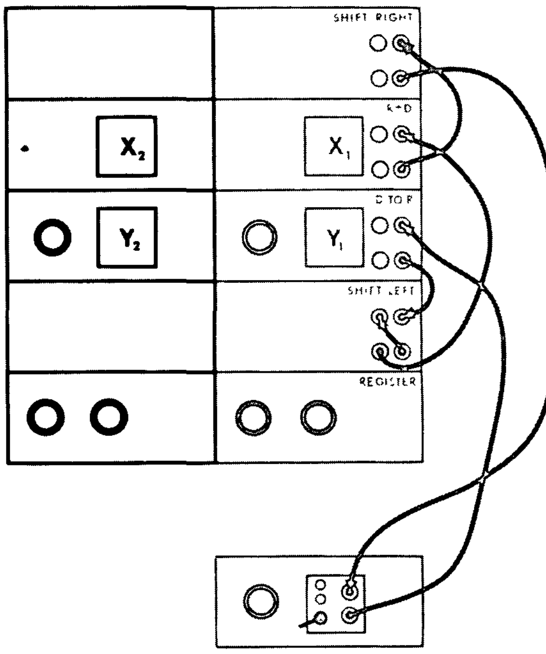


Figure 14

In Figure 14, the REGISTER manifold of Figure 13 has been extended to process 24-bit numbers by installing additional units in laterally adjacent cells. Extender Overlays are mounted on appropriate Faceplate Boxes; the Electronics Packages in the left column of the manifold match those to the right. Note that the control cabling is unchanged. Two additional Parameter Blocks are required to hold the most-significant halves of the parameters X and Y (X_2 and Y_2).

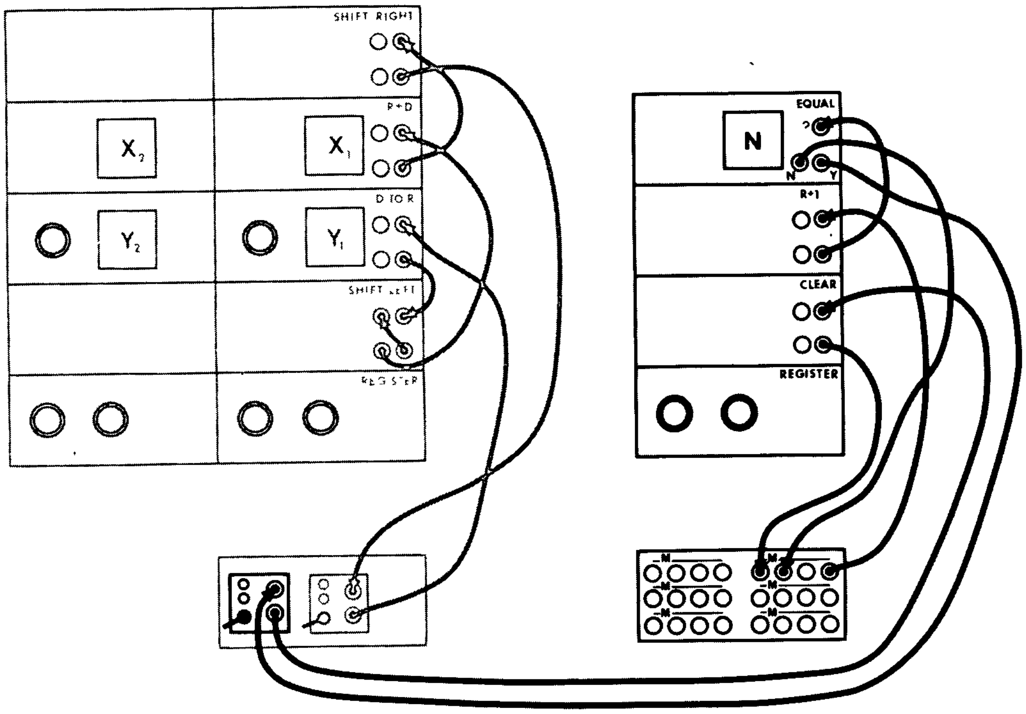


Figure 15

Leaving the manifold and control structure of Figure 14 intact for the moment, we proceed next to build a second, quite independent REGISTER manifold as in Figure 15, together with a MERGE unit and a new control network. This new structure acts as a counter. Flipping the leftmost Miniconsole switch causes a lengthy sequence of steps to occur: first the new REGISTER is cleared (the CLEAR Overlay on a LOAD unit Faceplate Box specifies this function), after which a completion signal is sent to one of the two inputs of a MERGE element (the remaining five elements are unused) whose output next evokes the function R + 1 to R (as determined by an Overlay for the ARITHMETIC unit), which is then followed by a test for equality to the parameter N (as determined by the EQUAL Overlay for the COMPARE unit). There are two alternative outcomes following this comparison, and a control signal will continue along one of the two paths as shown. If the 12-bit number in the REGISTER has not yet reached the value N set in the Parameter Block, the comparison completion signal will return via the N (No) output to the second of the MERGE element inputs (to "merge" with any signal on the first input in OR combination) and another MERGE output signal is then generated to repeat the sequence. The sequence will be repeated until the "count" held in the REGISTER equals the number N, whereupon the comparison completion signal is returned along the Y (Yes) pathway to the Miniconsole and all further action stops. We say that the "control loop" thus defined is traversed, or iterated, N times following the initial CLEAR operation.

Figure 16 shows the addition of another 24-bit REGISTER manifold to the structure of Figure 15, interconnected to the first 24-bit manifold in such a way as to "accumulate" the result

$$K + \sum_{i=1}^{N-1} \frac{1}{2} (X + 4i)$$

in REGISTER T. The heavy lines represent Data Cables, each carrying 12 bits of data from a REGISTER output connector to an input connector of another unit. The cable from REGISTER I carries the number i to the rightmost D TO R unit of the S-manifold, while the cables from the S REGISTER carry the sum ½(X + 4i) to the R + D data inputs of the T-manifold for accumulation with previous results. Since the parameter X and the number i are both 12 bits in length, zeroes are supplied to the most-significant (leftmost) data inputs of the S-manifold as shown. The control loop defined in Figure 15 has been rearranged to include the required S- and T-manifold operations, and the reader will have no difficulty in tracing the sequence of steps initiated by flipping the Miniconsole switch.

It is easy to see how the process of Figure 16 can be further extended to include the generation or retrieval of an *indexed* value, X_i, by including within the control loop some additional macromodular process whose structure in Figure 17 is simply designated by the box labelled FIND X_i. The result now left in REGISTER T is the sum

$$K + \sum_{i=1}^{N-1} \frac{1}{2}(X_i + 4i).$$

Finally, the entire calculation of Figure 17 can be made a subsequence which can be "called" just as a subroutine would be "called" in programming practice. Figure 18 shows the replacement of the manually-operated Miniconsole by a CALL unit through which three independent (though of course non-simultaneous) calls can be made along three separate pathways which themselves might appear in some further superstructure not shown. The result of the calculation, Y, appears on the T REGISTER output cables.

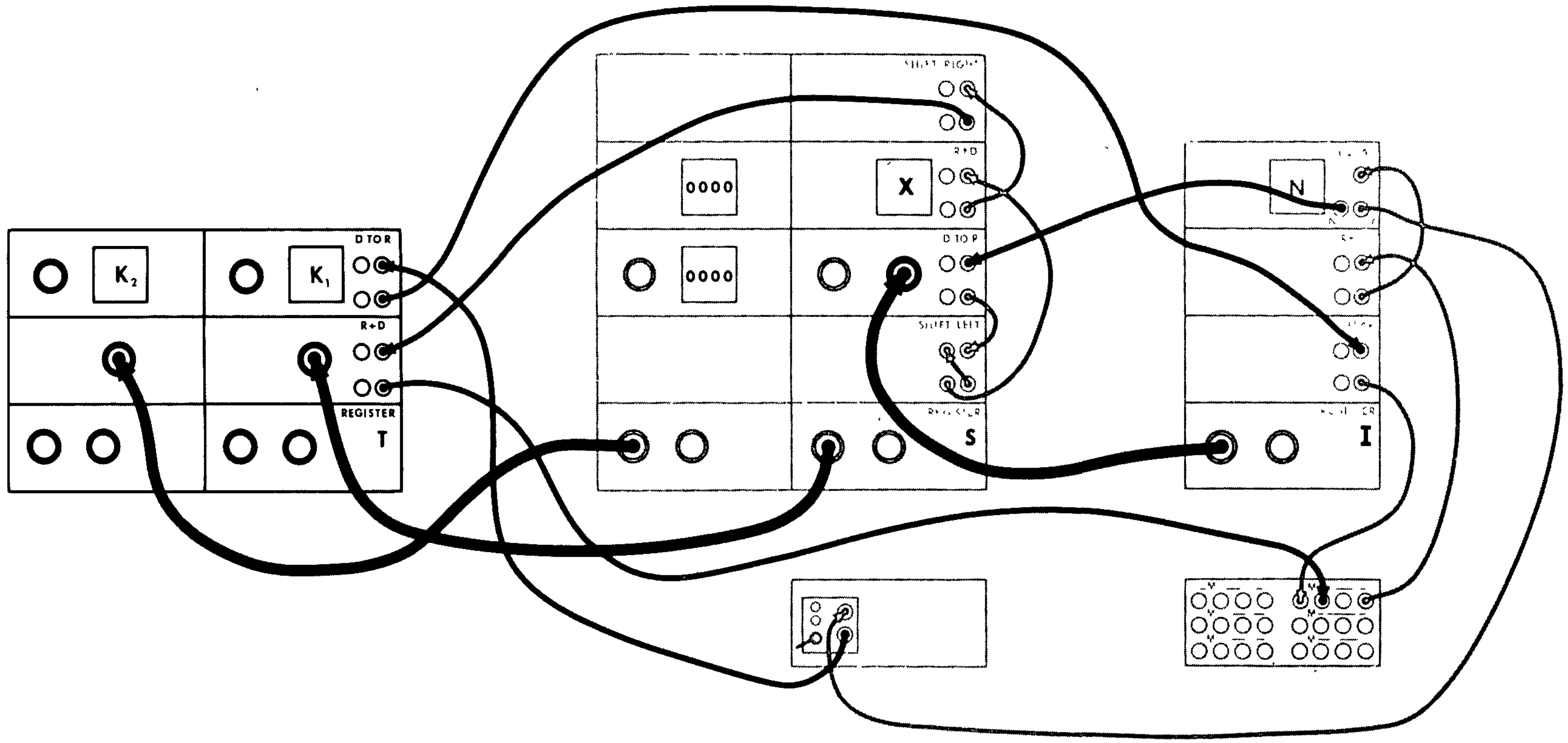

K2
K1
D TO R
R+D
REGISTER
T
0000
0000
X
SHIFT RIGHT
R+D
D TO P
SHIFT LEFT
REGISTER
S
N
I

Figure 16

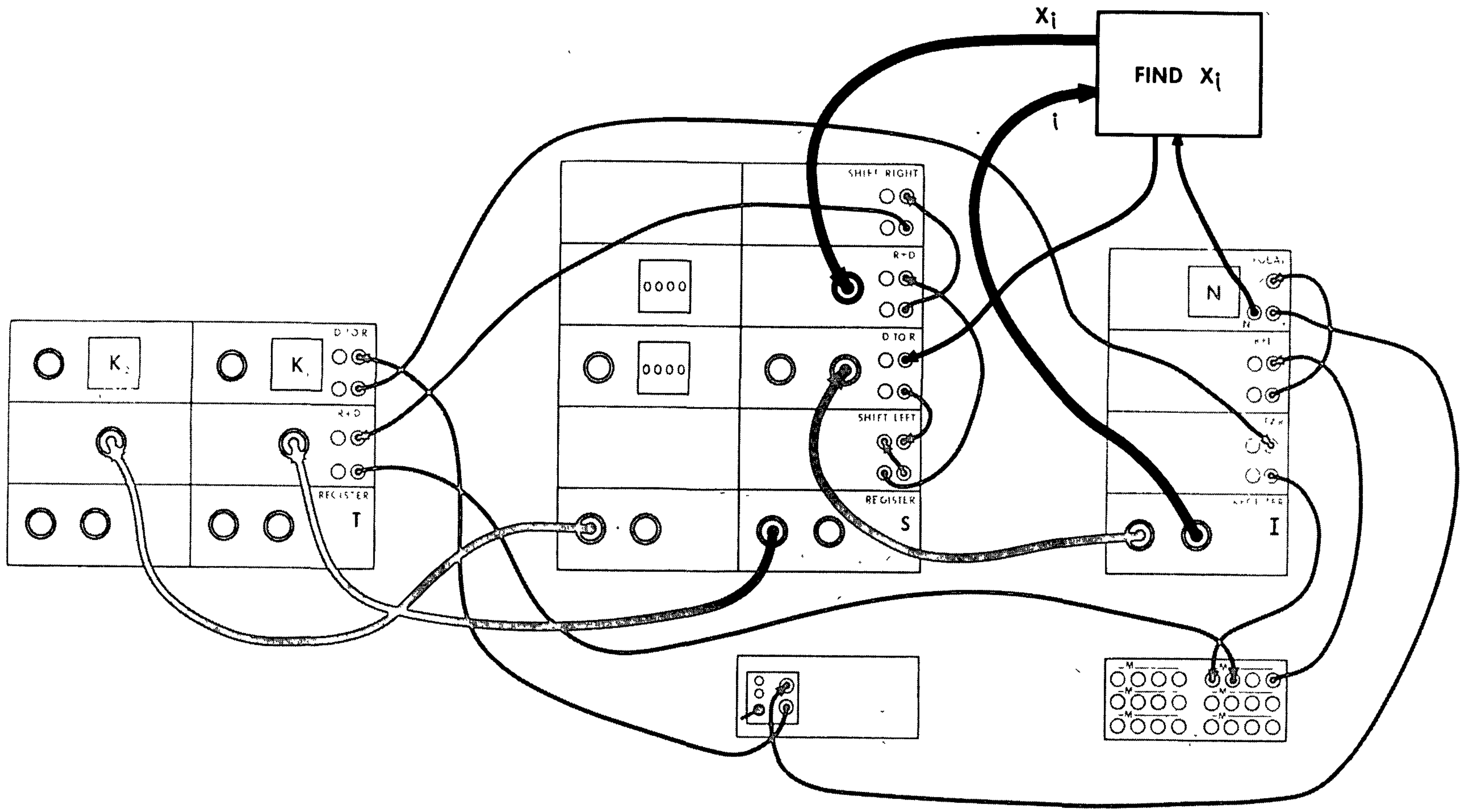
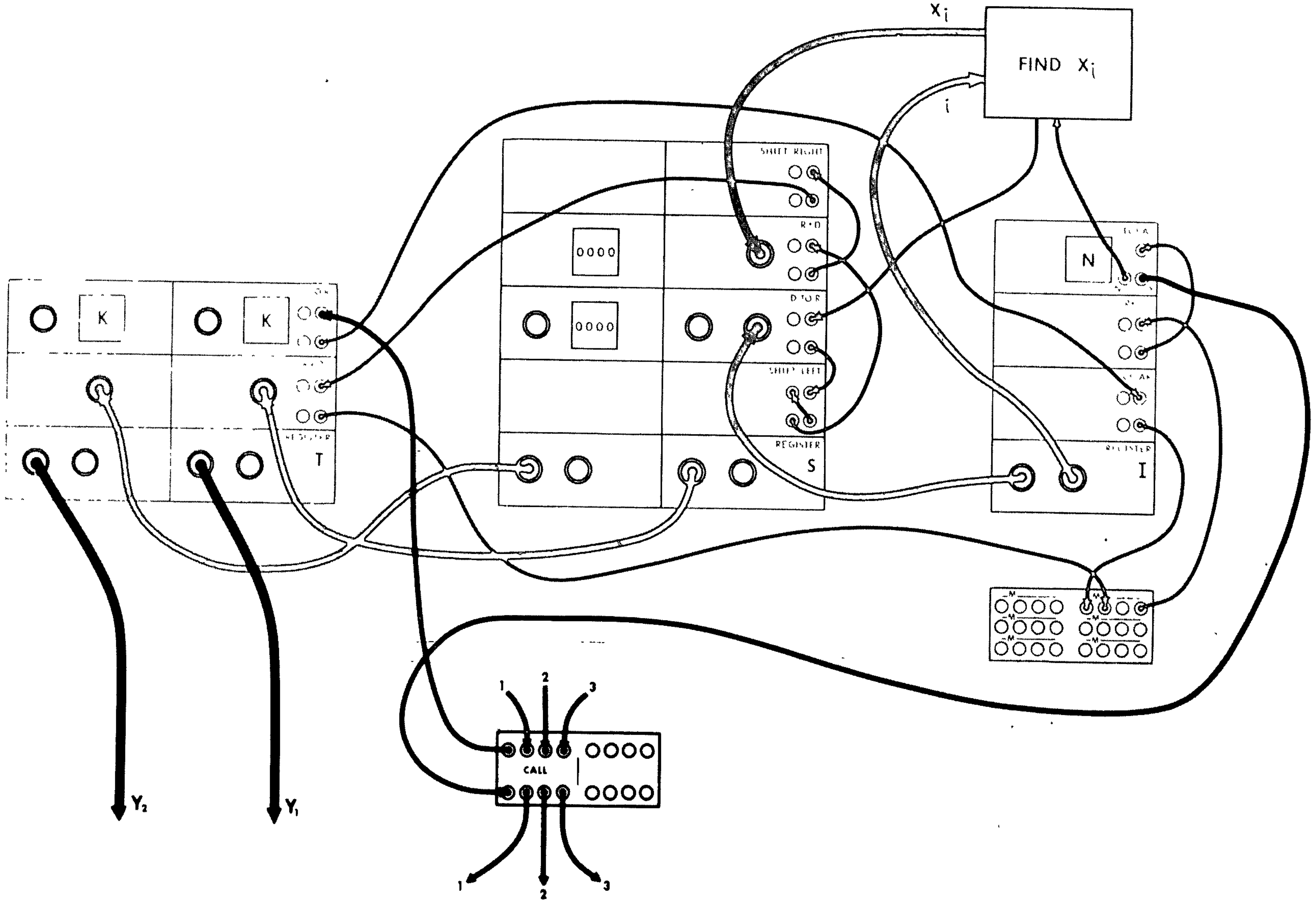

X_i
FIND X_i
i
SHIFT RIGHT
R+D
D TO R
SHIFT LEFT
REGISTER
S
0000
0000
K_2
K_1
T
N
I

Figure 17



x_i
FIND x_i
i
SHIFT RIGHT
R+D
D+OR
SHIFT LEFT
REGISTER
S
0000
0000
K
K
T
N
I
CALL
1
2
3
Y_2
Y_1

Figure 18

EXAMPLES OF SYSTEMS

To illustrate macromodular systems that have been applied to biomedical research problems, three specific macromodular systems that have been built and used will be described. The first, the CHASM, is a simulation and modeling system that is convenient for carrying out lengthy calculations at high speed. It is presented in considerable detail and illustrates the relatively small amount of documentation needed to completely describe a macromodular system of moderate size and complexity. The second, MMS-4, is an example of the use of macromodules in an evolving system that has grown in five years from a program on a MicroLINC with 8K of 12-bit memory to a very powerful and highly accessible system for modeling and displaying molecule structure. The third, the ARGUS Preprocessor, is an example of part of an electrocardiographic monitoring system designed and built in a short time in response to an immediate need which could not be met by other means

CHASM

The CHASM (*CH*arlie, *A*ntharvedi, and *S*evero's *M*achine) is a macromodular computer which is designed to compute certain probabilities associated with a class of Markov processes. This class of random processes provides useful models for the mechanisms that generate spike discharges in certain neurons in the central nervous system. The algorithm used is based upon one originally implemented as a LINC program (Molnar, 1966). The CHASM version described here is similar to that presented in a preliminary description in 1967 (Molnar, Ornstein, and Anné, 1967), but incorporates design improvements and modifications resulting from changes in the functional definition of macromodules from that assumed earlier. A number of errors in the earlier description are also corrected here.

The Markov process analyzed by the CHASM is a continuous-time Markov process, with both continuous and discontinuous transition mechanisms, that has been studied by Kolmogorov (1931) and Feller (1936), and more recently described by Takacs (1962). The rationale for the use of this type of Markov process as a neural model is discussed by Stein (1964), Molnar (1966), and Molnar and Pfeiffer (1968). The model assumes that a neuron receives inputs in the form of discrete events generated by an inhomogeneous Poisson random process with rate $\rho(t)$. The state of the neuron is represented completely by its depolarization, D. Each input event instantaneously increases the value of D by a positive amount, R(D). During the intervals between input events, the depolarization decays monotonically toward zero at a rate $\alpha[D] = \frac{dD}{dt}$. Whenever D reaches or exceeds a threshold value T, an *output* event is generated and the value of D is reset to an initial value $D_0 \geqslant 0$.

These assumptions, subject to certain conditions on $\alpha[D]$ and $R[D]$, define a class of Markov processes within which particular members are distinguished by specifying $\rho(t)$, R(D), T, D_0 and decay function $\alpha[D]$. Such a Markov process is completely characterized by the transition probability distribution, $F[t,x \mid \tau, \nu]$, which is the probability that the state variable D is less than x at time t, conditional upon D having the value ν at a previous time τ. Although it is known that $F[t,x \mid \tau, \nu]$ must satisfy certain integro-differential equations (Molnar, 1966, p.131), analytic solutions for these equations are not known for most cases of interest as neural models. Such a solution, if available, would yield $F[t,x \mid 0,D_0]$, the probability distribution for the depolarization at time t given an initial value of $D = D_0$ at $t = 0$.

Instead we can recursively calculate $F[(k+1)\Delta t,x \mid 0,D_0]$ from $F[k\Delta t,x \mid 0,D_0]$ using the Chapman-Kolmogorov equation (Cox and Miller, 1965, p. 205):

$$(1) \qquad F[(k+1)\Delta t,x \mid 0,D_0] = \int_0^T F[(k+1)\Delta t,x \mid k\Delta t,\nu] \; d_\nu F[k\Delta t,\nu \mid 0,D_0]$$

We can approximate $F[(k+1)\Delta t, x \mid k\Delta t, \nu]$ for small Δt as follows The probability of n events in the time interval $[k\Delta t,(k+1)\Delta t)$ is given by the Poisson distribution:

$$P_n(k\Delta t) = \frac{e^{-\lambda_k} \lambda_k^{\,n}}{n!}$$

where $$\lambda_k = \int_{k\Delta t}^{(k+1)\Delta t} \rho(t)dt$$

Thus: $$P_0(k\Delta t) = 1 - \lambda_k + o(\lambda_k),$$

$$P_1(k\Delta t) = \lambda_k + o(\lambda_k),$$

and $$P_n(k\Delta t) = o(\lambda_k) \qquad \text{for } n > 1.$$

For small λ_k and small Δt, we ignore terms $o(\lambda_k)$, assume that a jump can occur only at $t = k\Delta t$, and obtain the approximation:

$$F[(k+1)\Delta t, x \mid k\Delta t, \nu]$$

$$= 0, \qquad x \leqslant \nu - a[\nu]\Delta t;$$

$$= 1 - \lambda_k, \qquad \nu - a[\nu]\Delta t < x \leqslant \nu + R(\nu) - a[\nu + R(\nu)]\Delta t,$$

$$= 1, \qquad \nu + R(\nu) - a[\nu + R(\nu)]\Delta t < x \leqslant T.$$

Applying the Chapman-Kolmogorov equation (1), and adopting abbreviated notation:

$$F[k\Delta t, x] \equiv F[k\Delta t, x \mid 0, D_0],$$

$$F[(k+1)\Delta t, x] = \int_0^T F[(k+1)\Delta t, x \mid k\Delta t, \nu] \; d_\nu F[k\Delta t, \nu]$$

$$F[(k+1)\Delta t, x] = \int_0^{\nu_0(x)} 1 \; d_\nu F[k\Delta t, \nu] + \int_{\nu_0(x)}^{\nu_1(x)} (1 - \lambda_k) \; d_\nu F[k\Delta t, \nu] + \int_{\nu_1(x)}^{T} 0 \; d_\nu F[k\Delta t, \nu]$$

(2) $$F[(k+1)\Delta t,x] = \lambda_k F[k\Delta t, \nu_0(x)] + (1 - \lambda_k) F[k\Delta t, \nu_1(x)],$$

where $\nu_0(x)$ satisfies the equation $x = \nu_0 + R(\nu_0) - a[\nu_0 + R(\nu_0)]\Delta t$ and can be interpreted as that value of ν from which a *jump* followed by a *decay* transition would lead to the new value x at a time Δt later; and where $\nu_1(x)$ satisfies the equation $x = \nu_1 - a[\nu_1]\Delta t$ and can be interpreted as that value of ν from which a *decay* transition would lead to the new value x at a time Δt later.

A 4096 x 24 macromodular memory segmented into four 1024-word quadrants is used as the working storage for the CHASM. The first quadrant, addresses 0-1777 octal, is used to store $F[k\Delta t,x]$ for odd values of k, and the second quadrant, addresses 2000-3777 octal, for even values of k. Values of $F[k\Delta t,x]$ are stored as non-negative 24-bit signed fractions, with the largest possible value, 3777 7777 octal, taken as the approximate representation of one. By suitable scaling, x may be allowed to take on the values 0, 1, 2, 3, ..., 1023 and T be fixed equal to 1023 without loss of generality.

Tables representing the transition functions $\nu_1(x)$ and $\nu_0(x)$ are stored in the third and fourth memory quadrants respectively, with the left 12 bits interpreted as the integer part and the right 12 bits as the 2's complement of the fractional part of the tabulated functions. These functions, as well as the initial values for $F[0,x \mid 0,D_0]$ are loaded from an external source, which also provides successive values of λ_k as needed. This support is presently provided by a Spear MicroLINC 300, equipped with a macromodule interface, which also reads results from the CHASM and initiates each calculation for successively increasing values of k.

The following description of the CHASM is detailed enough to define the structure of the system. Various points, such as the placement of data cables, are conveyed implicitly, but in a manner sufficient to define a working system functionally (i.e. leaving only choices to be made at assembly time which do not endanger correctness of the implementation of the desired algorithm, although they may influence the speed). In this sense, the documentation is complete.

Figure 19 shows a photograph of the front face of the CHASM, while Figure 20 identifies the electronics package type and overlay type for each cell (by the code number shown in the lower right) and the data cable connections (by the names associated with operations)

The CHASM registers and their major functions are as follows:

A	(24 bits) serves as a general accumulator for 24-bit arithmetic and communication with the LINC interface
B	(24 bits) serves as a 24-bit buffer and result register for multiplications.
C	(12 bits) contains the multiplier during multiplication.
D	(24 bits) serves as a display buffer.
E	(12 bits) serves as a scaling step counter for display.
M	(12 bits) controls memory quadrant selection and address selection.
S	(12 bits) is the memory address register and a buffer for loading C.

The LINC interface contains a 24-bit output register L, which can be read into A, and a 12-bit output register λ, which communicates with S. The LINC interface can also read data from A, and controls initiation and senses completion of four macromodular operation sequences.

The control structure is defined by the flow diagrams of Figures 21 and 22 in which the operations identified in each box correspond to individual macromodular operations or to macromodular subroutines. Four different operation sequences can be called by the LINC.

LOAD transfers a 24-bit word from the LINC output to a memory location specified by the contents of the S register, reads the new contents of memory back into A for check purposes, and increments S.

MEMCHK loads the A register from the memory location specified by the interface λ-register.

INITIALIZE sets the contents of the S and M registers to zero.

EXECUTE carries out one complete calculation of $F[(k+1)\Delta t,x]$ from $F[k\Delta t,x]$, taking λ_k from the interface λ register.

The major calculation of the CHASM, carried out by EXECUTE, involves straightforward evaluation of equation (2) for successively increasing values of x. First, $\nu_1(x)$ is obtained by table look-up. Next, $F[k\Delta t,\nu_1(x)]$ is obtained by linear interpolation in the table for $F[k\Delta t,\nu]$. Next, multiplication by $(1 - \nu_k)$ is carried out and the result temporarily saved in memory. The process is repeated to obtain $\lambda_k\nu_0(x)$, with the final result

$$(1 - \lambda_k)F[k\Delta t,\nu_1(x)] + \lambda_k F[k\Delta t,\nu_0(x)]$$

stored in memory location x. A sense switch on the console selects optional display of the calculated value of $F[k\Delta t,x]$ as a function of x, with a variable scale factor selected by a parameter switch in (Switch$\rightarrow$).

The CHASM is an evolving system that has been used and modified extensively (Arthur and Molnar, 1971). The particular version shown here represents a system in transition, and contains several sub-optimal arrangements, vestiges of earlier forms, concessions to debugging convenience, and even a CALL module not connected to anything. The assembly and check-out of the system typically requires a day, and proper construction of the driving tables for $\nu_0(x)$ and $\nu_1(x)$ and correct interpretation of the results have been more of a hindrance than the physical realization of the system itself. The CHASM, as described here, can carry out thirteen repetitions of the EXECUTE operation per second, compared with an execution rate of four per second for a less general and less precise algorithm executed on the MicroLINC, and sixty-seven executions per second for the algorithm programmed in Fortran and executed on a CDC-6600. A newer version of the CHASM, using a prototype MULTIPLY macromodule, carries out about 26 executions per second. Another version of the CHASM adapted to model neurons with inhibitory as well as excitatory inputs has been built and operated, but the performance has not yet been fully evaluated.

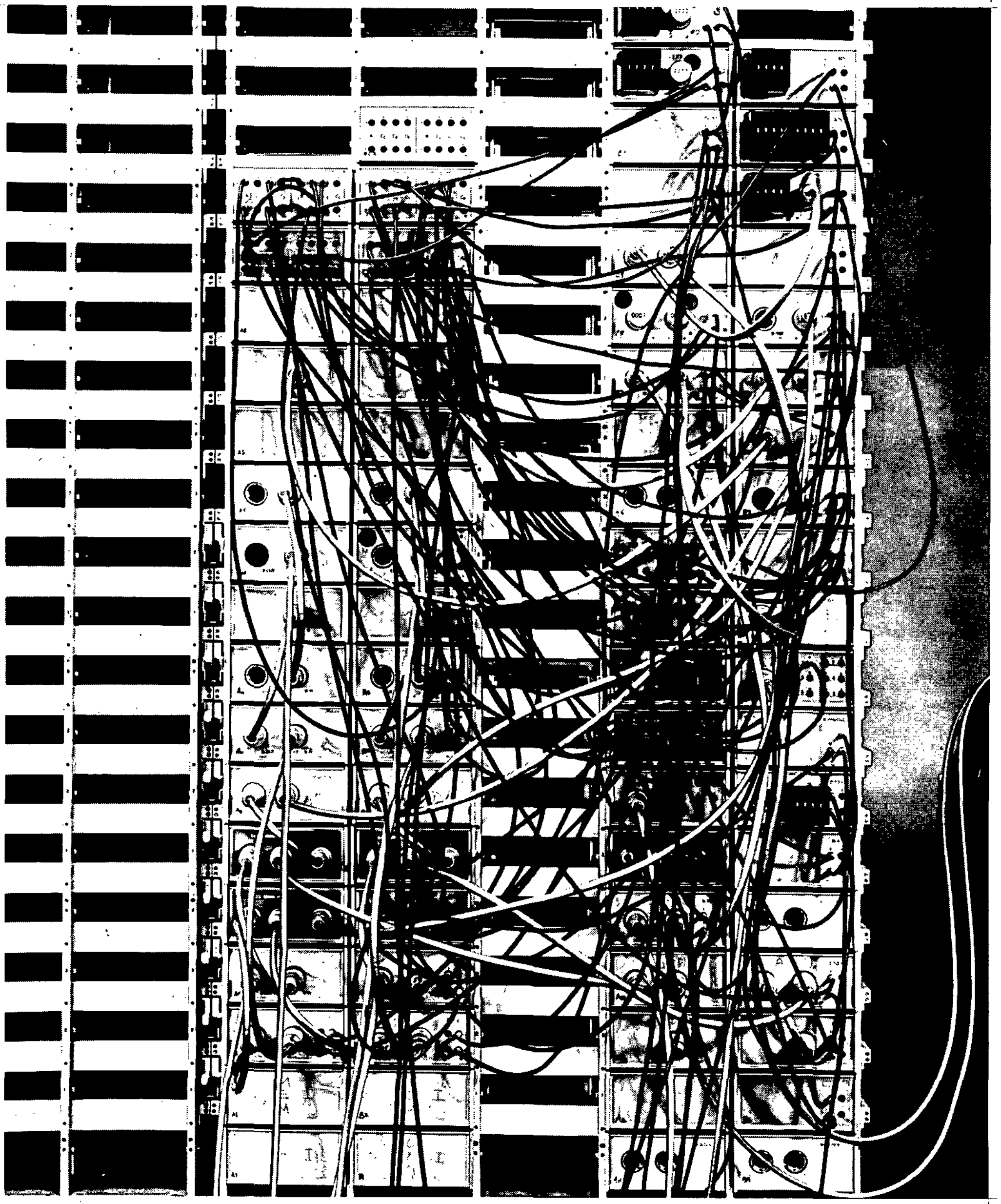
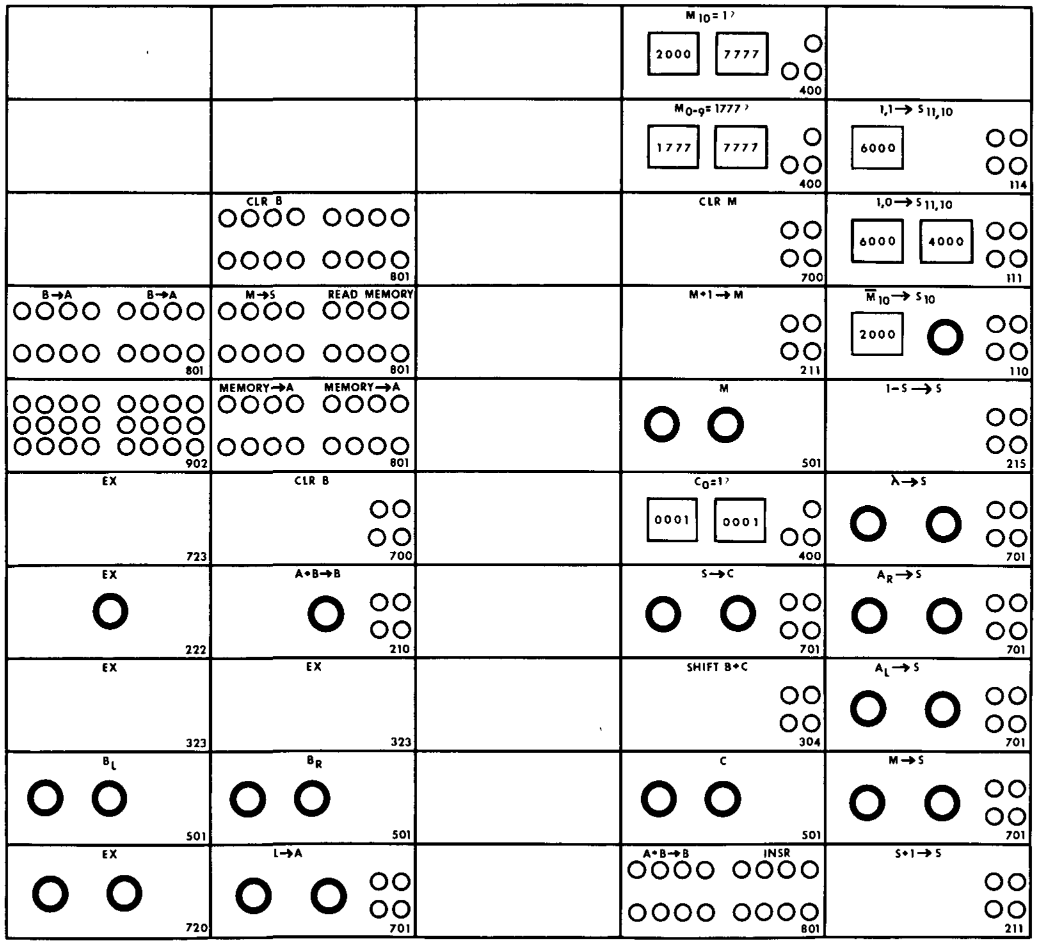


Figure 19

The macromodular frame assembly of the CHASM (*CH*arlie, *A*ntharvedi and *S*evero's *M*achine). Identification of the function of each cell is given in Figure 20, while the control structure is described in Figures 21 and 22. Cables leaving the area shown in the photograph connect to a console (to the right) and to the LINC interface (underneath).


M10=1?
2000
7777
400
M0-9=1777?
1777
7777
400
1,1→S11,10
6000
114
CLR B
801
CLR M
700
1,0→S11,10
6000
4000
111
B→A
B→A
801
M→S
READ MEMORY
801
M+1→M
211
M̄10→S10
2000
110
902
MEMORY→A
MEMORY→A
801
M
501
1-S→S
215
EX
723
CLR B
700
C0=1?
0001
0001
400
λ→S
701
EX
222
A+B→B
210
S→C
701
AR→S
701
EX
323
EX
323
SHIFT B+C
304
AL→S
701
BL
501
BR
501
C
501
M→S
701
EX
720
L→A
701
A+B→B
INSR
801
S+1→S
211

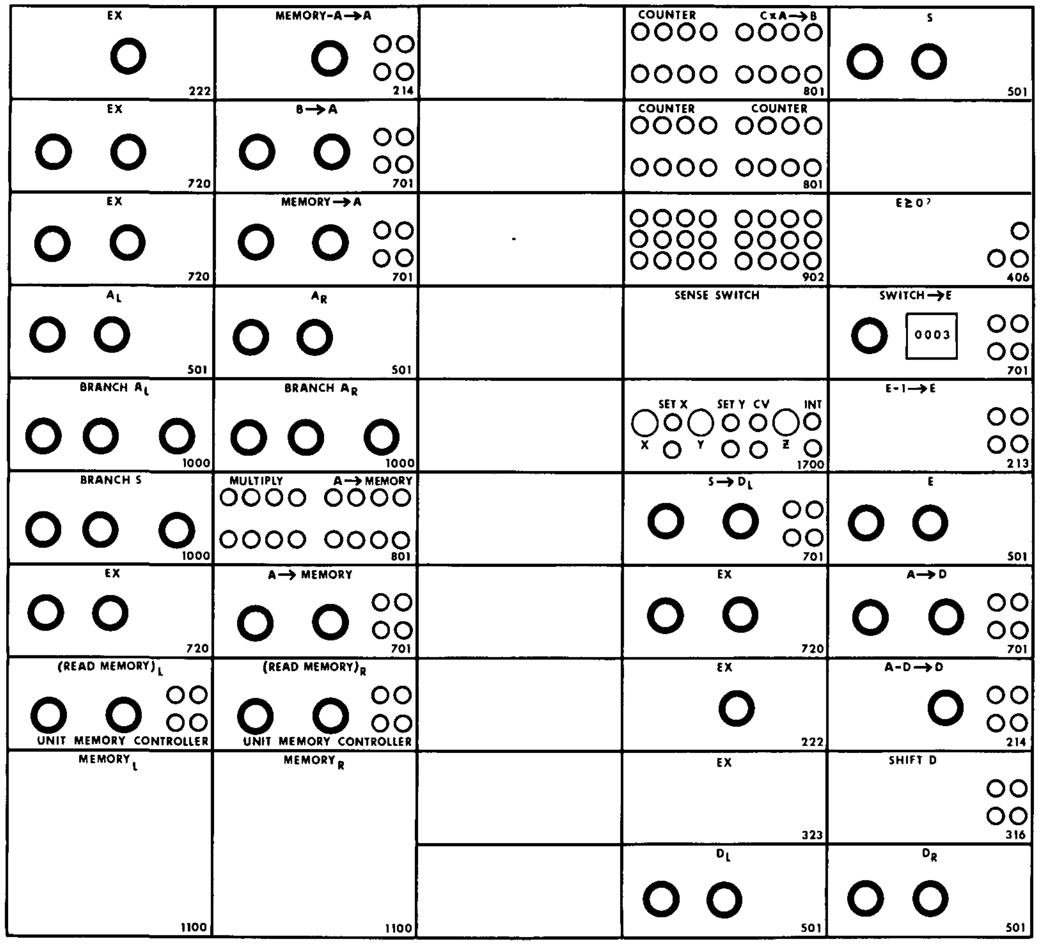


Figure 20

A cell map of the CHASM system pictured in Figure 19 identifies modules within the frame. Notations on each overlay assign names to operations and identify the module type and overlay code by the number in the lower right. Parameter switch locations and values are shown, and the locations of Data Cables are implied by the operation and register names. The names on CALL modules (code 801) identify subroutines or operations that are called. The CALL modules identified as COUNTER call the multiply kernel 12 times as explained in the text.

The abbreviation EX denotes a module that extends in word length the operation of the module to the right. Other abbreviations used are: CLR - clear, CV - convert, INT - intensify, $( )_L$ - left half-word, $( )_R$ - right half-word, INSR - interpolate normalize subroutine.

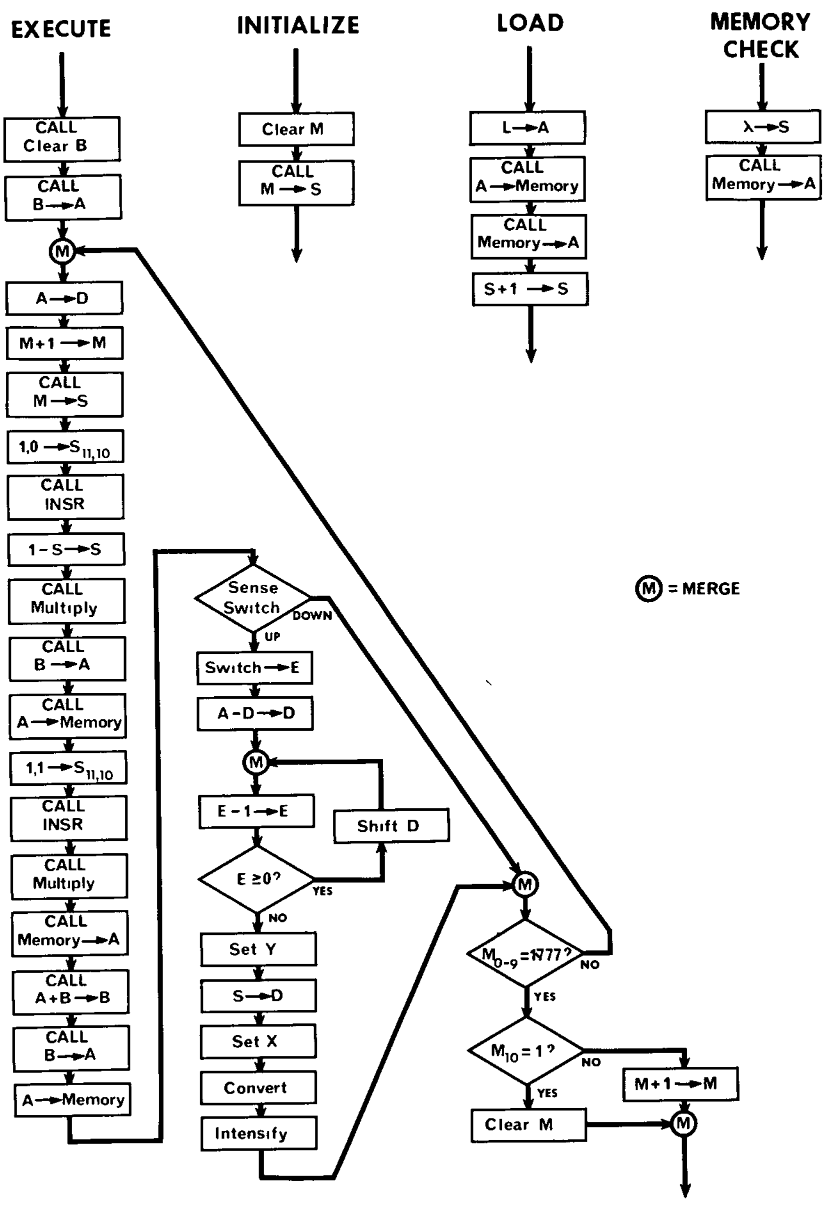


Figure 21

Control flow diagram for the four CHASM programs. Each program is initiated by a control signal from the LINC interface, and in turn communicates its completion back to the LINC. Control terminals associated with individual operations can be identified on the cell map of Figure 20.

CALLS

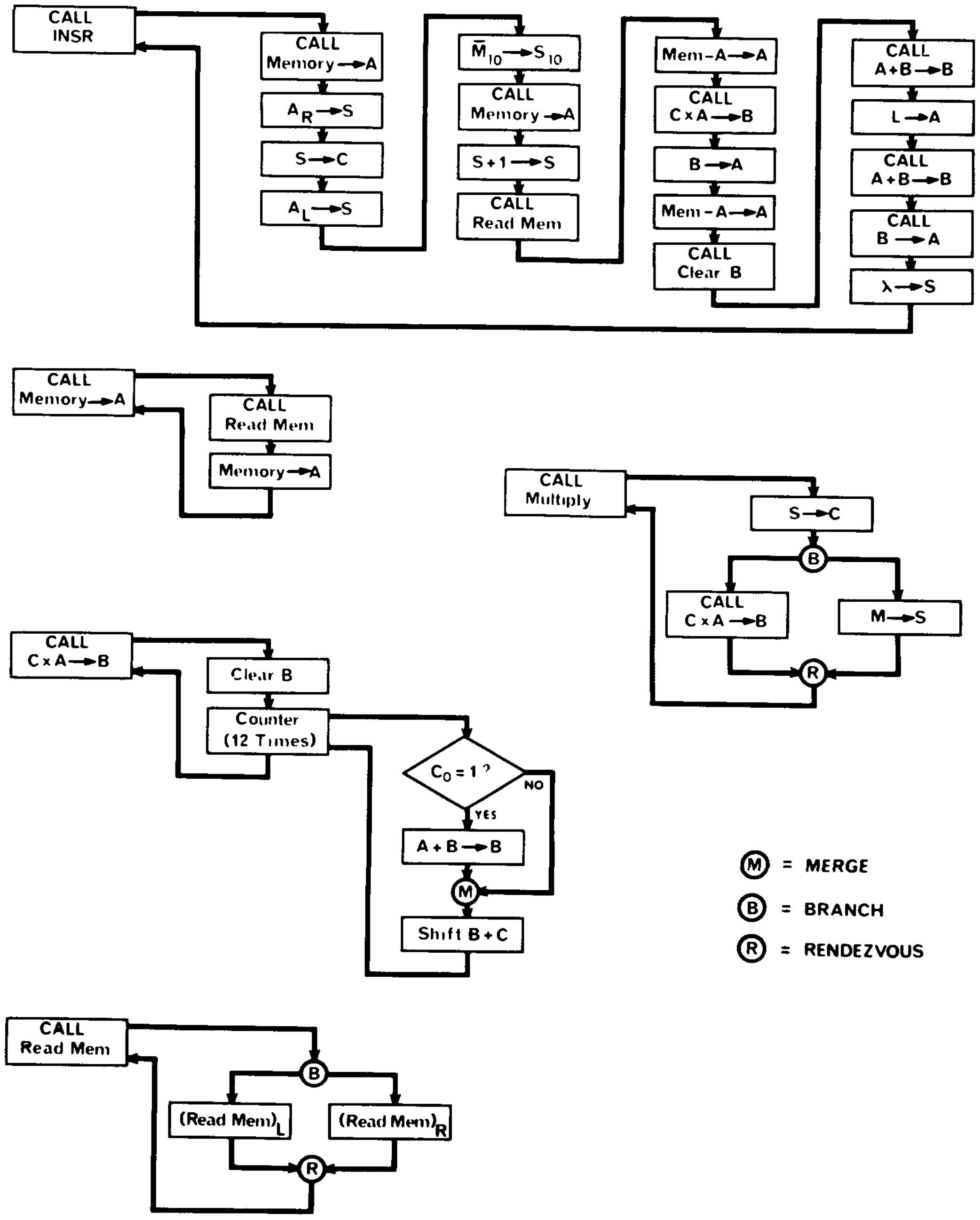


Figure 22

Subroutines used in the CHASM. Nesting of subroutines and concurrent execution of operations are direct and easy to realize. Calls used to increase the number of accesses to single operations are not shown.

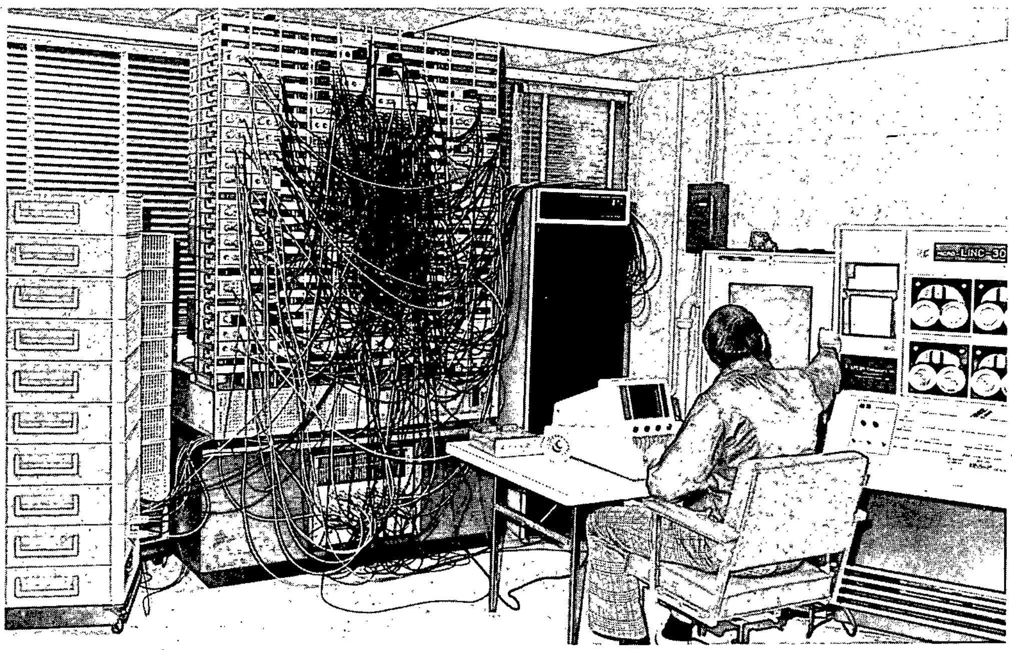


Figure 23

The MMS-4 system as of October, 1972. From left to right in the rear can be seen: macromodule-compatible memory stacks; the macromodular frame assembly; the Evans & Sutherland LDS-1 Matrix Multiplier and Line-Drawing Scope; the Spear MicroLINC 300. The LINC interface can be seen below the frame assembly. The LINC console in the foreground and a set of control knobs provide the user with means to manipulate the operation of the system.

MMS-4

The MMS-4 system, shown in Figure 23, is a powerful and convenient system for manipulation and display of molecular models. It is an interesting example of an evolving system which includes a general-purpose stored program computer (Spear MicroLINC 300), and commercially built specialized hardware (Evans and Sutherland Line Drawing Scope and Matrix Multiplier). About 150 macromodules implement specialized data handling and control functions optimized for molecular modeling, and serve to tie together the other parts of the system in a functionally convenient and efficient way. The macromodules effectively implement powerful high-level commands which are called by the MicroLINC. For example, one complete molecule picture is generated in response to a single LINC command. The thousands of individual steps required are controlled by "soft wired" control pathways established by macromodular control cables. The system can, in present form, comfortably handle large protein models such as myoglobin (Figure 24), and can carry out such functions as continuous rotation of a complete molecule about a desired axis, geometric transformation of part of a molecule with respect to another part, and fitting of molecular models to simultaneously displayed contour plots of experimentally determined electron density maps.

This system has evolved from early experiments using the MicroLINC alone (Barry, et al., 1971), (Dickson, et al., 1972), on which display, manipulation, and data storage techniques were developed and applied to small molecules. Systems MMS-1, MMS-2, MMS-3 (Ellis, Fritsch, Dodds, 1971), representing successive experiments in structuring a system for convenient and flexible extension of the LINC-based system, had lifetimes of a few weeks each. The present version, MMS-4 (Ellis, et al., 1974), has been in continuous existence for over a year, and has slowly evolved through weekly-to-monthly additions and modifications. The most recent change was an increase in the precision of coordinate storage and manipulation from 12 bits to 24 bits. This change, made in response to the needs of a visitor, required 18 additional modules. The hardware modification and changing of the programs was completed with less than a one-day interruption in the use of the system. The software for the system has also evolved in a smooth manner. Important functions have been implemented one at a time in macromodules and moved out of the LINC without major disruptions to system use.

The direction and speed of evolution of the macromodular molecular modeling system have been determined principally by the needs and ideas of the vigorously active group of system user/designers, whose work is described in detail elsewhere (Marshall and Bosshard, 1972) (Marshall, Barry, et al., 1972) (Marshall, Beitch, et al., 1972) (Marshall, Bosshard, et al., 1972). Major accomplishments have been the construction of a library of protein structures from coordinates supplied by others, the generation of several films showing various aspects of protein structure (Beitch, et al., 1971), and the development of techniques for fitting models to X-ray maps of electron density.

The next major step in this evolution is the implementation in macromodules of operations which determine and test the distances between any pair of atom centers. The resulting system will be used to enumerate exhaustively sterically possible conformations of biologically interesting molecules (Bosshard, et al., 1972), a task that is overwhelming for existing computer systems. Experimentally determined parameters such as nuclear magnetic resonance spectra or circular dichroism spectra can then be compared with those calculated for each sterically allowed conformation (Barry, North, et al., 1971).

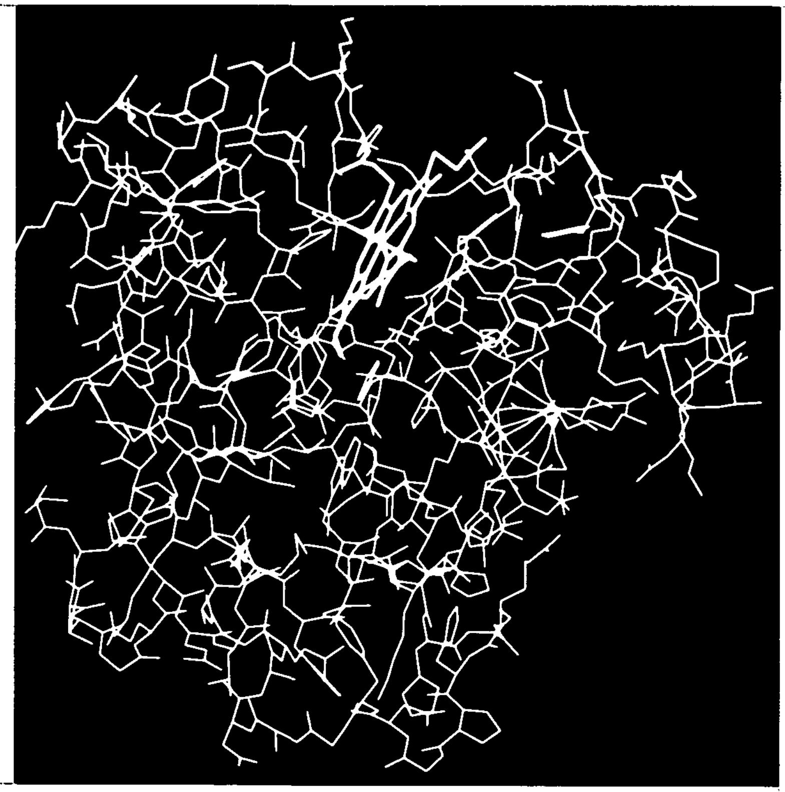


Figure 24

The structure of myoglobin (Watson, 1969) as photographed from the MMS-4 scope. Lines connect the positions of linked atom centers in the structure. Superimposed electron density maps can be displayed simultaneously.

ARGUS Preprocessor

The development at Washington University of effective algorithms for electrocardiographic monitoring in coronary care units (ARGUS - *AR*rhythmia *GU*ard *S*ystem [Cox, Fozzard, Nolle, Oliver, 1968] [Nolle, Clark, 1971] [Nolle, 1972]) has led to interest in the processing of taped electrocardiograms obtained from ambulatory patients. Tens of thousands of hours of recording are necessary from hundreds of subjects to obtain statistically meaningful answers to each of a number of interesting questions in the study of the relation of heart rhythm to coronary artery disease. A high-speed system for analyzing these recordings is clearly required, perhaps with the capability of processing an hour's worth of electrocardiogram in a minute, a 60 to 1 gain in speed.

In the spring of 1971 the need for such a high-speed processing system emerged clearly, but the feasibility turned on a technical question of computer processing speed Could a system be designed that would handle an input data rate of about 300,000 bits/sec, given the 27 memory cycles required, on the average, to process each bit? An experiment was devised that concentrated on the preprocessing section of ARGUS. If this preprocessor, called AZTEC (*A*mplitude-*Z*one, *T*ime *E*poch *C*oding), could keep up with the input data, the reduction in data rate that AZTEC could provide would make the task of succeeding portions manageable.

Since the preprocessing algorithm (Cox, Nolle, Fozzard, Oliver, 1968) was well defined, the design of a macromodular system to accomplish the task was a matter of only a few hours. The next day, the system was assembled and debugged (Figure 25) Taking slightly under fifty modules, including modules used as debugging aids, the system was of modest size, but could remove from the stored-program portion of the system more than two-thirds of the memory cycles required by the ARGUS algorithms (Cox and Logue, 1971).

The performance was well in excess of that required: data presented at a megabit rate could still be processed, an increase in speed of 200 to 1 over real-time processing. Of course, some paths through the preprocessing algorithms take substantially more time than others (ranging from 2 μsec to 10 μsec). The performance of the system cited above is based on the worst-case path.

A comparison of the AZTEC preprocessor implemented in software and in macromodules is instructive (Cox, 1968), the most frequently used path through the algorithm takes the following times: CDC 6600 - 7 μsec, IBM 360/75 - 10 μsec, IBM 360/65 - 12 μsec, SIGMA 7 - 17 μsec, PDP 10 - 18 μsec, SIGMA 2 - 27 μsec, PDP 9 - 37 μsec. The equivalent path through the macromodular system takes 3 μsec. Little of the sophisticated processing hardware of the larger systems applies since the operations are largely composed of additions, tests and branches. The cost per execution favors the SIGMA 2, nearly the smallest of the conventional systems, and a substantial cost premium (about two orders of magnitude) must be paid for the CDC 6600 to achieve only a factor of four increase in speed over the SIGMA 2. The macromodular system achieves a factor of 9 increase in speed with a much smaller cost premium (presently about a factor of two)

The success of the experiment with the ARGUS preprocessor has led us to pursue the development of a high-speed processing system for the analysis of electrocardiograms recorded from ambulatory patients. Succeeding portions of the ARGUS algorithm are being implemented in an IBM System 7. Tapes reproduced at 60 times real time will be played directly into the macromodular preprocessor whose output will provide the input for the System 7. Future experiments will help to shape the best form for this preprocessor. If the system proves to be successful scientifically, a further optimized preprocessor design may be developed, based upon the experience with the macromodular system.

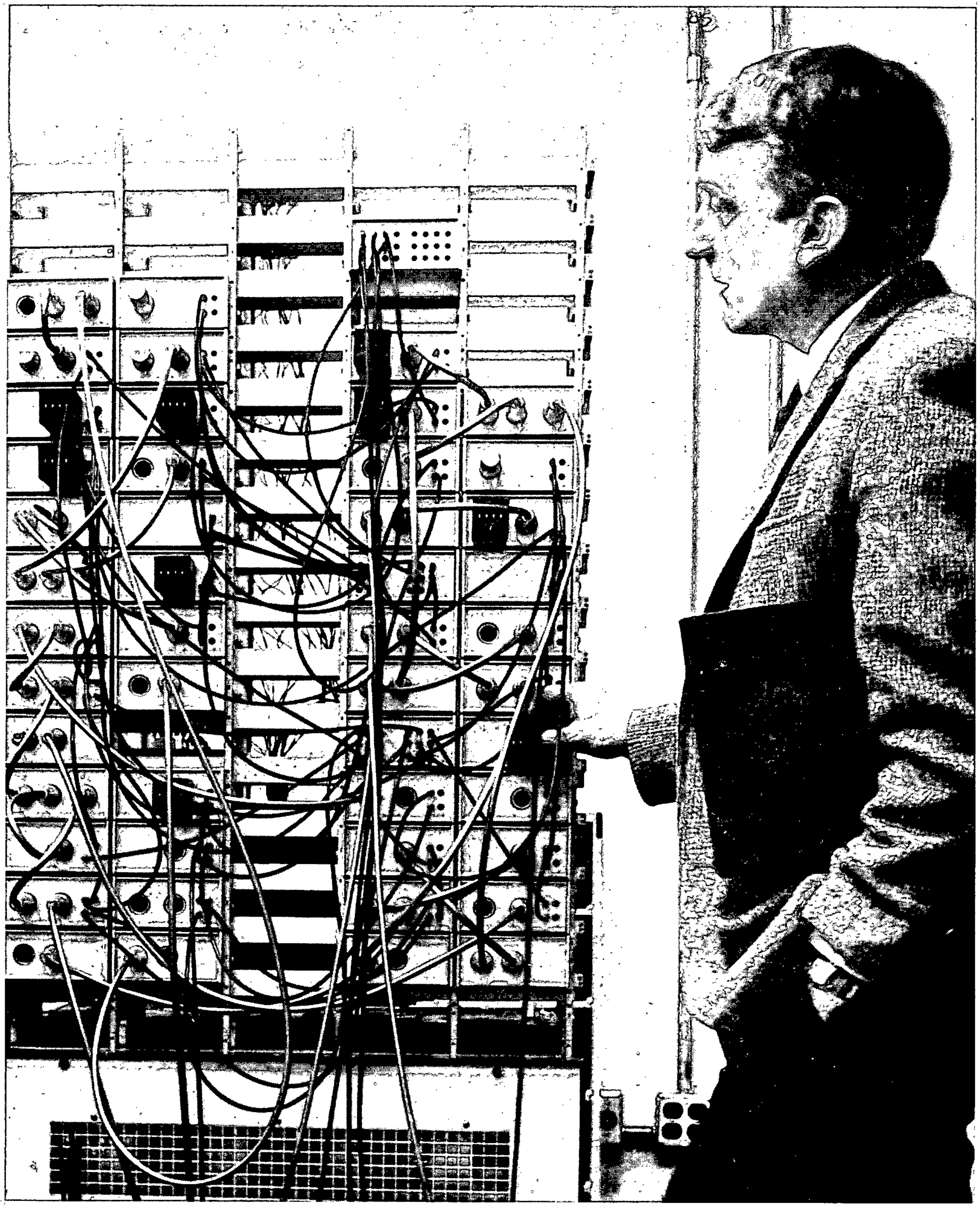


Figure 25

The AZTEC (ARGUS Preprocessor) System can accept input electrocardiographic data at speeds up to 200 times real time.

CONCLUSION

Although the process of learning to design, build, and use macromodules has been lengthy and difficult, we are encouraged by our recent experience to believe that the approach is technically and intellectually sound The ability to quickly and easily design and modify computer systems of novel structure and arbitrary size introduces a new degree of freedom of great value to the problem-solver Particularly significant is the increased ease of thinking about a problem in purely algorithmic terms while taking for granted the ability to construct a system which will efficiently execute that algorithm in a direct and simple manner.

Continuing research use of our macromodule inventory by ourselves and our colleagues at Washington University and elsewhere should teach us a great deal more about administering the shared use of a macromodular resource. We have yet to determine how to exploit optimally the inherent ecological soundness of macromodules which allows them to be recycled indefinitely.

Development of a "restructured" form of macromodules intended to preserve the essential features of macromodules in a more cost-effective and exportable form is now in an advanced stage, and engineering prototypes are being built. While different in many details, macromodules in this new form will, in general, have the same functional structure as the macromodules described here, and will allow the same style of use (Clark and Molnar, 1972).

Acknowledgements

The authors wish to express special thanks to those who contributed to the assembly and preparation of the material in this report. Mr John Matthews built and documented the CHASM version described, Mr. Robert Ellis and Dr. Garland R Marshall provided material on the MMS-4 system, and Dr. J. R. Cox, Jr contributed the description of the AZTEC system. Technical assistance in the preparation of the text and figures was provided by Miss Gerelyn Hollingsworth, Mrs. Mary Ann Kelly, and Mrs. Sandra Stewart, and photographic work was done by Mr David Shupe and Mr. Gary Parker.

APPENDIX

MACROMODULAR SYSTEMS BUILT DURING THE PERIOD
JANUARY 1971 THROUGH JUNE 1972

SYSTEM	CELLS	FUNCTION/DESCRIPTION
Memory Tester	18	Tests the 2-cell core memory macromodule.
DIS 1	27	A 24-bit stored-program special-purpose display processor programmed to display functions stored as tabulated values in its memory.
JANC	24	A 12-bit stored-program general-purpose computer with 4096-word 1-microsecond memory.
HISTO†	23	Displays selected segments of memory as histograms with parameter-switch control of bar width, scaling, and memory segment location.
Stimulus Generator	10	Stores samples of an arbitrary periodic function and delivers them as a continuous analog signal.
4-SORT	19	Sorts four 12-bit numbers by recursive execution of comparisons and exchanges of data in adjacent positions.
Pseudorandom Number Generator†	20	Based on an algorithm used in an early TX-0 program.
LISP Machine	77	A rough equivalent to PDP-1 LISP without I/O; i.e., a LISP interpreter plus a dozen or so primitive functions. Speed about the same as for PDP-10-compiled LISP.
Relaxation Technique	25	Performs a linear ordering by relaxation averaging.
3-D Histogram	30	Generates a 3-dimensional histogram and displays it on a macromodular display scope.
Pseudorandom Number Generator	24	Implementation of the algorithm described by Rader, Rabiner, and Schafer (Bell System Technical Journal, Nov. 1970). Computes and displays a histogram of the number distribution.
Hadamard Transform Processor	52	Implementation of an algorithm for Fast Hadamard Transforms. Operates about eight times faster (215 ms) than an IBM 360/50 programmed in Fortran IV.

†Designed and constructed by a visitor.

SYSTEM	CELLS	FUNCTION/DESCRIPTION
Gaussian Random Number Generator†	31	A special processor, based on the Rader algorithm described above, which forms a sum of 12 uniformly distributed random numbers to produce Gaussian numbers, and displays these as a bell-shaped frequency histogram.
Pair Interchange Sorter	30	Sorts a series of numbers using a pair interchange algorithm.
Pattern Occurrence Recorder	20	Examines selected field of data input stream and records the number of occurences of each pattern that appears.
Log Generator	23	Forms logarithm of n by an incremental method suggested by a time interval analog technique.
Stack Computer	47	An elementary general-purpose (programmable) stack computer.
Sorting System	97	An implementation of D.L. Shell's minimal storage sorting algorithm. Sorts up to 4096 twelve-bit words.
Digital Filter	56	A general recursive second-order digital filter with constant coefficients, to be used in ECG filtering experiments.
Macromodular Interlock	29	Designed to test the feasibility of a two-system interlock, in preparation for the construction of a multiprocessor.
FFT Computer*	85	Performs Fast Fourier Transforms. The number of complex 12-bit points transformed is any power of 2 up to 2^{12}; transform time is approx. 0.67 sec. for 4096 12-bit complex numbers.
Digital Oscillator*†	16	Solves the equations $y=x'/2^n$; $x=y'/2^m$. If the initial conditions cause arithmetic overflow, a type of limit cycle results, the period of which (for 12-bit words) can be as high as $(211{,}071{,}162)_8$ coordinate pairs before repeating.
P.D.E. Processor I P.D.E. Processor II	60 60	Solves a restricted class of partial differential equations (elliptical for I, elliptical and parabolic for II), using Monte Carlo methods.

*Assembled and demonstrated in Harriman, New York.

†Designed and constructed by a visitor.

SYSTEM	CELLS	FUNCTION/DESCRIPTION
CUDDLY I	124	Finds sociable, amicable, and perfect numbers, printing out each set as found. Reached the number 4,002,206 after 85 hours, using 24-bit arithmetic on chains no longer than 256, and appears to have found 3 new sets. The overflow cases remain to be resolved by larger systems.
Markov Algorithm Processor	60	Implementation of a Markov algorithm for string manipulation, controlled on-line from the keyboard. Uses both LINC and macromodular frame memory for storage.
Fabri-Tek Memory Tester	12	Used to test the Fabri-Tek core memory units used in many macromodular systems.
CHASM Multiplier CHASM	20 60-66	Developmental stages in the implementation of a Markov Process model of a single neuron. Recursively computes state probability functions from initial conditions and transition probability tables.
AZTEC QRS Detector ARGUS Preprocessor ECG Data Encoder	46 35 100 60	Stages in the implementation of a system for analysis of tape-recorded ECGs played back at 60 times real time. Uses an algorithm developed at George Washington University for QRS detection and boundary determination based on analysis of the first derivative of the ECG. Final data compression: 100 to 1.
MMS-1 MMS-2 MMS-3 MMS-4	85 70 81 116	Successive developmental versions of a system composed of an Evans and Sutherland Line Drawing System, a Spear Micro-Linc 300 computer, and up to 120 macromodules. The MMS-4 system allows rapid and flexible manipulation and display of molecules containing as many as 2500 atoms, and the fitting of protein models to electron density maps obtained by X-ray diffraction.
MARC I COMRAD MARC II MARC III	85 94 91 97	A series of stored program computers designed to determine essential concurrencies in a sequential computer program (implementation of Fisher's Algorithm). Successive versions were designed by using the current version to analyze and improve its own algorithm.
RAP-1	30	Provides on-line editing and retrieval of sounds. Macromodules are interfaced to M-compatible devices (devices that are not macromodules but are compatible with them): a microphone, A-to-D converter, storage disk, and clock.
Speech Pattern Analyzer	80	Uses a "real-time", Newton-Raphson-like algorithm to solve the parameters of a linear prediction model of speech.

SYSTEM	CELLS	FUNCTION/DESCRIPTION
Glitch Counter	32	Records histogram data on flip-flop glitch times.
MUMPS Character Processor	64	Implements the character processing routines of the MUMPS interpreter in macromodules, the rest of MUMPS in the PC-1200. Considerable speed increases over the full PC version were realized.
Matrix Inverter	91	Inverts a matrix by Gauss-Jordan elimination. Constructed as a class project.
Speech Processing Display Controller	64	A fully buffered driver for a high-speed display scope, which forms part of an interactive computer system for speech analysis and synthesis.
HOBBIT	68	Performs the calculations necessary to determine all possible ring conformations of a molecule.
Basilar Membrane Model	74	Simulates the Kim non-linear phenomenological model for basilar membrane motion.
SIMBC Aggregate	133	Simulates a broadcast computer system, displays the results, and stores the optimal ordering obtained so far.
Video Data Acquisition System	16	Accepts video data generated by an A-to-D converter, and stores it in real time for later processing. The system is presently in use at the cardiac care unit of Jewish Hospital in St. Louis.
Digital Stimulus Generator	37	Produces three phase-locked tones for use in experiments on combination tone perception.
Broadcast Channel Tester	30	Tests the performance of a cable television channel, by supplying random inputs and checking the outputs.

REFERENCES

Arthur, R.M., and C.E. Molnar (1971) "Calculation of Neural Discharge Probability Using a Macromodular Computer System." Proceedings of the 24th Annual Conference on Engineering in Medicine and Biology, Las Vegas, Nevada. November. Vol. 15.8.

Barry, C.D , R A. Ellis, S.M. Graesser, and G.R. Marshall (1971). "CHEMAST: A Computer Program for Modeling Molecular Structures." Proceedings of the International Federation of Information Processing Societies, Ljubljana, Yugoslavia.

Barry, C.D., A.C.T. North, J.A Glasel, R.J.P. Williams, and A.V Xavier (1971). "Quantitative Determination of Mononucleotide Conformations in Solution Using Lanthanide Ion Shift Broadening NMR Probes." Nature,(London), Vol. 232, 236.

Beitch, J.J., R.A. Ellis, J.M. Fritsch, and G.R Marshall (1972). "A Protein Sampler." 16mm Film, Computer Systems Laboratory, Washington University, St Louis, Missouri.

Bosshard, H E., C D. Barry, J.M. Fritsch, R.A Ellis, and G.R. Marshall (1972). "BURLESK: The Use of Systematic Conformational Searches in Chemistry." Proceedings of the 1972 Summer Simulation Conference, San Diego, California, Vol. I, 581

Clark, W A., and C.E. Molnar (1972). "The Promise of Macromodular Systems." Digest of Papers, Compcon '72, San Francisco, California, 309-312

Cox Jr., J.R. (1968). "Economy of Scale and Specialization." Computer Design, Vol. 7, 77-80.

Cox Jr., J.R., H A. Fozzard, F M. Nolle, and G C. Oliver (1969). "Some Data Transformations Useful in Electrocardiography." *Computers in Biomedical Research,* Vol 3, Waxman and Stacy, Editors, Academic Press, New York.

Cox Jr., J.R., and R.D Logue (1971). "Some Observations on the Economics of Computer Systems for Monitoring Electrocardiographic Rhythms." Computers and Biomedical Research, Vol. 4, 447-459.

Cox, D.R., and H.D. Miller (1965). *The Theory of Stochastic Processes.* Wiley, New York, 205.

Cox Jr., J.R., F.M. Nolle, H A Fozzard, and G.C. Oliver (1968). "AZTEC, A Preprocessing Program for Real-Time ECG Rhythm Analysis." IEEE Transactions on Biomedical Engineering, Vol. BME-15, No. 2, April, 128-129.

Dickson, C., C.D. Barry, R.A. Ellis, J.M. Fritsch, and G.R. Marshall (1972). "A User's Guide to CHEMAST." Technical Memorandum No. 156, Computer Systems Laboratory, Washington University, St. Louis, Missouri.

Dickson, C. (1974). "A Macromodule User's Manual," Technical Report No. 25, Computer Systems Laboratory, Washington University, St. Louis, Missouri.

Ellis, R.A., J.M Fritsch, and C.B Dodds (1971). "Molecular Modeling System-1." Technical Memorandum No. 120, Computer Systems Laboratory, Washington University, St. Louis, Missouri.

Ellis, R.A., J.M. Fritsch, T.H. Jacobi, and G.R. Marshall (1974) "The Macromodular Modeling System (MMS-4)." Technical Memorandum No 174, Computer Systems Laboratory, Washington University, St. Louis, Missouri

"Index to Technical Memoranda." (1972). Computer Systems Laboratory, Washington University, St. Louis, Missouri.

"Index to Technical Reports." (1972). Computer Systems Laboratory, Washington University, St Louis, Missouri.

Kolmogorov, A. (1931). "Über die Analytischen Methoden in der Wahrscheinlichkeitsrechnung." Math. Annalen, Vol. 104, 415-458.

Marshall, G.R., C.D. Barry, H.E. Bosshard, and N. Eilers (1972). "Effects of Methyl for Hydrogen Substitution on Peptide Backbone Conformations " Abstracts of the 16th Annual Biophysics Society Meeting, Vol 12, 158.

Marshall, G R., J. Beitch, R.A. Ellis, and J.M Fritsch (1972). "Macromodular Modeling System: The Insulin Dimer " Diabetes, Vol. 21, Supp. 2, 506.

Marshall, G R , and H.E. Bosshard (1972) "Angiotensin II: Studies on the Biologically Active Conformation." Circulation Res. Vol. 31, Supp. 2, 143.

Marshall, G.R , H E. Bosshard, N. Eilers, and P. Needleman (1972) "Constraints on the Receptor-Bound Conformations of Angiotensin II " Chemistry and Biology of Peptides, Ann Arbor Science, Ann Arbor, Michigan.

Molnar, C.E. (1966) "Model for the Convergence of Inputs upon Neurons in the Cochlear Nucleus." Doctor of Science Dissertation, Massachusetts Institute of Technology, Cambridge, Massachusetts.

Molnar, C.E., S.M Ornstein, and A. Anné (1967). "The CHASM: A Macromodular Computer For Analyzing Neuron Models." Spring Joint Computer Conference, AFIPS Proceedings, Vol. 30, Thompson Books, Washington, D.C., 393-401.

Molnar, C.E., and R.R Pfeiffer (1968). "Interpretation of Spontaneous Spike Discharge Patterns of Neurons in the Cochlear Nucleus." Proceedings of the IEEE, Vol. 56, 993-1004.

Newell, J.A. (1974). "Data and Control Signal Distribution in a Macromodular Data-Processing Manifold." Technical Report No. 21, Computer Systems Laboratory, Washington University, St Louis, Missouri.

Nolle, F M. (1972). "ARGUS, A Clinical Computer System for Monitoring Electrocardiographic Rhythms." Doctor of Science Dissertation, Washington University School of Medicine, St. Louis, Missouri.

Nolle, F.M., and K.W. Clark (1971). "Detection of Premature Ventricular Contractions Using an Algorithm for Cataloging QRS Complexes." Proceedings of the San Diego Biomedical Symposium, Vol. 10, 85-97.

Ornstein, S.M., M.J. Stucki, and W.A. Clark (1967) "A Functional Description of Macromodules " Spring Joint Computer Conference, AFIPS Proceedings, Vol. 30, Thompson Books, Washington, D.C. 337-355.

Stucki, M J., S.M. Ornstein, and W.A. Clark (1967). "Logical Design of Macromodules." Spring Joint Computer Conference, AFIPS Proceedings, Vol. 30, Thompson Books, Washington D C., 357-364.

Takacs, L. (1962). *Stochastic Processes.* Methuen, London.

Watson, H.C. (1969) "The Stereochemistry of the Protein Myoglobin." *Progress in Stereochemistry,* Vol. 4, Aylett and Harris, Editors, Butterworth, London, 299-333.

Unclassified
Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1 ORIGINATING ACTIVITY *(Corporate author)* Computer Systems Laboratory Washington University St. Louis, Missouri	2a. REPORT SECURITY CLASSIFICATION Unclassified 2b. GROUP

3 REPORT TITLE

OVERVIEW OF MACROMODULES

4 DESCRIPTIVE NOTES *(Type of report and inclusive dates)*
Final Report 4/1/65 through 12/1/73

5 AUTHOR(S) *(First name, middle initial, last name)*

Christine D. Coaker, Editor

6 REPORT DATE February, 1974	7a. TOTAL NO OF PAGES 140	7b. NO OF REFS 44

8a. CONTRACT OR GRANT NO DOD (ARPA) Contract SD-302	9a. ORIGINATOR'S REPORT NUMBER(S) Volume I of Part 1
b. PROJECT NO ARPA Project Code No. 655	
c.	9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*
d.	Technical Report No. 44

10 DISTRIBUTION STATEMENT

Distribution of this document is unlimited.

11 SUPPLEMENTARY NOTES	12 SPONSORING MILITARY ACTIVITY ARPA - Information Processing Techniques, Washington, D.C.

13 ABSTRACT

This volume contains a Foreword to the Final Report, two excerpted reprints that present the initial conception of macromodules and means for implementing them as seen in the early days of the project, and a third reprinted report that presents a summary and overview as of the autumn of 1972.

DD FORM 1 NOV 65 1473 (PAGE 1)
S/N 0101-807-6801

Unclassified
Security Classification

Unclassified
Security Classification

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Macromodular Computer Systems						
Modular Computer Systems						
Macromodular Approach to Computer Design						
Functional Description of Macromodules						
Logical Design of Macromodules						
Macromodular System Design						
Restructurable Computer Systems						
Special-Purpose Computer Systems						

DD FORM 1 NOV 65 1473 (BACK)
(PAGE 2)

Unclassified
Security Classification