ENGINEERING RESEARCH INSTITUTE
UNIVERSITY OF MICHIGAN
ANN ARBOR

THE LOGICAL DESIGN OF AN IDEALIZED GENERAL-PURPOSE COMPUTER

ARTHUR W~~.~~alter BURKS

IRVING M. COPI

Project 1828

BURROUGHS CORPORATION
RESEARCH CENTER
PAOLI, PENNSYLVANIA

30 October 1954

Éпсм
1
итк
1837

Table of Contents

Preface

This is the second (and last) report of a series of which "Theory of Logical Nets" [2] was the first. We shall make some comments about the relations of the two reports.

"Theory of Logical Nets" gives a foundational analysis of logical nets (diagrams of digital computing circuits) in general. This second report presents a particular logical net, which represents the structure of an idealized general-purpose digital computer. Thus the first report contains the foundations for this second one, which presents an application of the nets analyzed in the first report.

The logical design of the general-purpose computer presented here was worked out by the first author independently of Project 1828. It was this work that stimulated the research already embodied in the first report. Since this design shows how the nets of the first report may be used, it seemed desirable to present the logical design of the computer as a second report in the series. In doing so we found it necessary to work out a set of logical formulas describing the behavior of the computer, and, since this material is of interest as an application of symbolic logic to computers, we have embodied it in the present report.

THE LOGICAL DESIGN OF AN IDEALIZED GENERAL-PURPOSE COMPUTER[1]

1. Introduction.

There are many known complete logical designs for general purpose digital computers, but all of them are in terms of specific systems of equipment and most of them are very complex. That complexity is enforced not entirely by logical but also by engineering considerations pertaining to the physical problems involved in the actual construction of a machine. It seems desirable, therefore, to present a design for a complete general-purpose digital computer which shows its basic logical structure in abstraction from the engineering problem of realizing that structure physically. We give such a design in the present paper, patterned in a general way after the machine described in [1] . Since we use some of the techniques of symbolic logic in its exposition, our logical design is much more "logical" than is usually implied by the use of that term.

The present paper is intended to serve two important purposes. First, as has been said, it is intended to show what is logically required for a general-purpose computer as distinguished from what is required physically, the latter being relative to the present state of the computer art. Second, it is intended by abstraction from the requirements of equipment to show how a design can be obtained which is much simpler, though admittedly idealized, and hence much easier to understand. Our presentation should have pedagogical value for communicating the basic structure of a general-purpose machine to those acquainted with the rudiments of symbolic logic but lacking in engineering background.

The present paper is a continuation of [2] , which gives a foundational analysis of logical nets (diagrams of digital computing circuits) in general. The structure of our idealized general-purpose computer is here represented by a particular kind of net studied in [2] , the well-formed net.

2. The Computer and its Parts.

Our automatic computing machine contains three main parts with communication channels between them (Fig. 1). The first part is the ARITHMETIC UNIT, which performs the arithmetic operations of instructions 4, 5, 6, 7, 8, and 9 of Fig. 2, and which "senses" the end digits of a number when that is required for the execution of instructions 13 or 14.

The second part of the computer is its STORAGE, whose bins contain both numbers to be operated on and instructions to perform specified operations. The STORAGE contains two parts, a serial storage and a parallel storage. The parallel storage contains 4,096 storage bins: bin 0, bin 1, bin 2, ..., bin 4095, whose numbers constitute the *addresses* of the bins.

The machine must provide a path along which numbers can be transferred between the STORAGE and the ARITHMETIC UNIT when instruction 4, 5, 6, 7, 10, or 11 is executed. In our computer this path is the *trunk line* $\underline{t}$, whose sixteen components $\underline{t}_0$, $\underline{t}_1$, ..., $\underline{t}_{15}$ are represented by horizontal lines connecting the ARITHMETIC UNIT and the STORAGE in the lower part of Fig. 1.

For the machine to solve a problem it must be given a list of specific instructions, whose execution will result in the solution to the problem. The execution of these instructions is under the direction of the CONTROL, the third main part of the computer, which consists of all the equipment of Fig. 1 not yet described. Instructions for the machine are stored in order in bins of the parallel storage, and, except when it is instructed to "jump" (see 12, 13, and 14 of Fig. 2), the computer executes these instructions seriatim. To this end count must be kept of the instructions as they are executed; this function is performed by that part of the CONTROL called the ADDRESS COUNTER, which at any given time will contain the address of the bin storing the next instruction to be executed.

A set of instructions for the machine to execute in solving a problem is a *routine*, and every routine, however complex, is constructed out of a small number of primitive or basic instructions (the fourteen listed in Fig. 2). As each instruction emerges from the STORAGE it goes to the CONTROL, and that part of the CONTROL called the OPERAND DECODER then stimulates the appropriate *control wires* to make the other parts of the computer perform whatever functions are required for the execution of that instruction. Eleven control wires lead from the CONTROL to the other parts of the machine: six, bearing the general label $\underline{A}$, from the OPERAND DECODER to the ARITHMETIC UNIT; and five, labeled $\underline{S}$, from the OPERAND DECODER to the STORAGE. There are also two control wires, labeled $\underline{C}$, which connect different parts of the CONTROL, both $\underline{C}_r$ and $\underline{C}_t$ lead from the OPERAND DECODER and CONTROL CLOCK to the ADDRESS COUNTER, and $\underline{C}_t$ leads also to the equipment in the center of Fig. 1 labeled $\underline{t}''_1$. Different basic instructions cause the CONTROL to stimulate different sets of control wires, as indicated in Fig. 2.

How these various parts perform their functions will be explained in detail: the STORAGE in Section 4, the ARITHMETIC UNIT in Section 5, and the CONTROL in Section 6.

3. Logical Symbolism and Diagrams.

Since our diagrams are logical nets in the sense of [2], the lines in them represent wires. Each net wire in our diagrams is in one of two *states* (0,1) at each discrete moment of time $\tau = 0, 1, 2, \ldots$. (Whenever we use τ as a time variable, it ranges over discrete moments 0, 1, 2, ... unless otherwise stipulated.) The two states 0 and 1 are complementary; they correspond to the truth values false and true, respectively. In our diagrams control wires are labeled with capital letters, other wires (information wires) are labeled with small letters from the Latin or Greek alphabets. To assert that the wire labeled $\underline{f}$ is activated or stimulated at time τ we write either $\underline{f}$, $\underline{f}(\tau)$, $\underline{f} = 1$, or $\underline{f}(\tau) = 1$; and to assert that it is inactive or not stimulated we write either $\underline{f} = 0$ or $\underline{f}(\tau) = 0$. Thus the symbols used to label wires are systematically ambiguous: any such symbol $\underline{f}$ is on the one hand a label for the wire beside which it appears, and on the other hand it is a function symbol $\underline{f}(\tau)$ whose argument τ ranges over successive discrete moments of time and whose values are the wire states 0 and 1.

Our primitive logical symbol, the stroke function *$f \mid g$*, means <u>not both</u> $\underline{f}$ <u>and</u> $\underline{g}$, and the corresponding diagrammatic *stroke element* appears in Fig. 3a as a square nucleus with two **input** **wires** (f and g) and one **output** **wire** $f \mid g$ (as in [2]). Other logical symbols we shall use are: negation, $\bar{f}$ (or alternatively $\sim f$), meaning *not* f; (inclusive) disjunction, $f \vee g$, meaning f *and/or* g; conjunction, fg (or alternatively $f \cdot g$), meaning f *and* g; material implication, $f \supset g$, meaning *not* f *or* g; equivalence, $f \equiv g$, meaning f *and* g *or* *not* f *and* *not* g; and inequivalence, $f \not\equiv g$, meaning f *and* *not* g *or* g *and* *not* f. Just as these six functions can be constructed out of the stroke function (see pp. 255-256 of [4]), so the six corresponding diagram elements can be constructed out of stroke elements, as shown in Figs. 3b, c, d, e, f, and g.[2] Each right-hand diagram in Figs. 3b, c, d, f, and g is a definitional abbreviation of the diagram to its left. Having introduced the symbol for negation, we have additional ways to assert that wire f is inactive at time τ: $\bar{f}$, $\bar{f}(\tau)$, $\bar{f} = 1$, and $\bar{f}(\tau) = 1$. Similarly we could assert that it is active by writing $\bar{f} = 0$ or $\bar{f}(\tau) = 0$.

The inequivalence element has many properties which make it very useful in computer work. It can be used as a **complementer**, for if either input is in state 1, the state of the output wire will be the complement of the state of the other input (this application was first pointed out in [7]). We shall later (in Section 5) make use of the following properties of inequivalence: it is both associative and commutative, and $\underline{f}_1 \not\equiv \underline{f}_2 \not\equiv \ldots \not\equiv \underline{f}_N$ is true just in case an odd number of its arguments are true. That it has these properties is readily proved by induction on the number of arguments. Hence we can have an inequivalence element with any number of inputs (see Fig. 7 for a 3-input inequivalence).

It is convenient also to introduce a generalized threshold element having $\underline{I}$ positive input wires $\underline{f}_1, \underline{f}_2, \ldots, \underline{f}_I$ and $\underline{J}$ negative input wires (a negative input wire to an element is one which attaches to that element through a negation element) $\underline{g}_1, \underline{g}_2, \ldots, \underline{g}_J$, and a positive integer threshold $\underline{T}$. Such an element is diagrammed in Fig. 3h, and its behavior is characterized by the equation $\underline{h} \equiv \left[\left(\sum_{i=1}^{I} \underline{f}_i + \sum_{j=1}^{J} \overline{\underline{g}}_j \right) \geqq \underline{T} \right]$ where Σ, $+$, and $\geqq$ have their usual mathematical significance.[3] For any $\underline{I}$, $\underline{J}$, and $\underline{T} \geqq 0$ the threshold element is easily constructed out of elements already available. Where $\underline{T} > \underline{I} + \underline{J}$, $\underline{h} = 0$ and the construction is trivial. The equation for the non-trivial case is an inclusive disjunction whose disjuncts are all possible conjunctions of $\underline{T}$ arguments from the set of $\underline{I} + \underline{J}$ arguments $\{\underline{f}_1, \underline{f}_2, \ldots, \underline{f}_I, \underline{g}_1, \underline{g}_2, \ldots, \underline{g}_J\}$, and hence may be realized by a net containing only negation, conjunction, and disjunction elements (see Theorem 12 of [2]). It should be clear that disjunction and conjunction elements with any number of input wires are special cases of our generalized threshold element.

A second primitive element used in representing digital computing circuits is the *delay element*, which appears in Fig. 3i as an oblong nucleus with one input wire f and one output wire g. As in the case of the other elements, each of its wires possesses one of the two states (0,1) at each discrete moment of time: the output wire g is in state 0 at $\tau = 0$, and thereafter it possesses the state possessed by the input wire at the prior moment of time. The equations which describe its behavior are: $g(0) = 0$ and $g(\tau + 1) \equiv f(\tau)$ for every time τ.

We use two additional primitive elements: an input *key* whose activation puts the wire attached to it into state 1, and an output *light* which emits a visible signal if and only if the wire attached to it is in state 1. These are diagrammed as circles with the letters K or L in their interiors, as in Fig. 4. They represent relatively slow ways in which the operator can insert or receive information from the computer.

We use only nets that are well-formed in the sense of [2], but to simplify our diagrams we sometimes omit arrowheads and wires, as between the small and large circles in Figs. 3e and 3g, and we sometimes show arrowheads going both ways to avoid duplication of wires. We also allow a *multiple junction*, that is, a confluence of $\underline{n} \geqq 2$ output wires, to abbreviate an $\underline{n}$-input disjunction element in the manner of Fig. 3j, whose behavior is governed by the equations $(\underline{f} \vee \underline{g} \vee \underline{h}) \equiv \underline{i} \equiv \underline{j} \equiv \underline{k}$. And we place a dot at the intersection of two lines to indicate that the wires represented by those lines are connected, as at the left of each right-hand key in Fig. 4. Our final convention is that (except for Fig. 3) wires having the same label are understood to be connected, whether they occur in the same diagram (as the $\underline{t}_i$'s in Fig. 7) or in different diagrams (as the $\underline{t}_i$'s in Figs. 1, 4, 5, and 7).

4. STORAGE.

4.1. Information.

At any one time a single wire can carry one *bit* of information, 0 in its off state, 1 in its on state. The two digits 0 and 1 suffice for the expression of all real numbers in binary notation, and the term "bit" is a contraction of "binary digit." The machine's vocabulary consists of *words*, each of which is a sequence of sixteen bits.[4] At any moment exactly one word will be on the trunk line, each of whose sixteen components $\underline{t}_i$ $(\underline{i} = 0, 1, 2, \ldots, 15)$ carries one of the word's sixteen bits. (Whenever we use the variable $\underline{i}$ as subscript of a letter which labels a wire, it ranges over 0, 1, 2, ..., 15 unless otherwise stipulated.) In the machine each word is an ordered set of wire states, and every word is expressed by a number in binary notation. These words can be used to express either numbers or instructions. Their use to express numbers will be explained in Section 5; in the present section we explain their use to express instructions.

A typical instruction, such as ADD X (4 of Fig. 2), is expressed by an instruction word whose first four digits $\underline{d}_0\underline{d}_1\underline{d}_2\underline{d}_3$ constitute its operand and whose rightmost twelve digits constitute its address ($\underline{X} = \underline{x}_4\underline{x}_5 \ldots \underline{x}_{15}$), which specifies the location of the storage bin containing the number to be operated on. (In the case of instructions 1, 2, 3, 8, and 9, the address part is unused.)

4.2. Cells and Storage Bins.

A cell, which is a circuit capable of receiving and storing one bit of information, is easily constructed out of the elements described in Section 3. To see how the cell at the top of Fig. 4 stores information, we first observe that $\underline{h}_0^n \equiv (\underline{R}_t^n \underline{t}_0 \vee \overline{\underline{R}}_t^n \underline{b}_0^n)$. Now consider what occurs at any time τ when all of the control wires are in state 0. If $\underline{b}_0^n(\tau) = 1$, its signal will pass through the conjunction element below $\underline{h}_0^n$ up to make $\underline{h}_0^n(\tau) = 1$, whence $\underline{b}_0^n(\tau + 1) = 1$, and if all control wires remain inactive, then $b_0^n(\tau+1) = 1$ will entail $h_0^n(\tau+1) = 1$ also, whence $b_0^n(\tau+2) = 1$. On the other hand, if $b_0^n(\tau) = 0$, no signal will reach h_0^n, and $h_0^n(\tau) = 0$, whence $b_0^n(\tau+1) = 0$, and, if all control wires remain inactive, then $b_0^n(\tau+1) = 0$ will entail $h_0^n(\tau+1) = 0$ also, whence $b_0^n(\tau+2) = 0$. Thus, if no control wires are activated during the interval in question, whatever state a cell possesses at time τ will continue to be possessed by it at times $\tau + n$ for $n = 1, 2, 3, \ldots$.

Note that each symbol b_i^n of Fig. 4 has a three-fold significance: it labels the output wire of the delay element to its left, it is a function symbol $b_i^n(\tau)$ as explained at the beginning of Section 3, and it also labels the cell within which it appears. This three-fold significance attaches to the labels of all cells within the machine: β_0^{-1} and β_0^1 in Fig. 6, a_i in Fig. 7, and α_i in Fig. 8.

Sixteen such cells are combined as in Fig. 4 to make one storage **bin**, which can receive and store a word, one bit in each cell. The **parallel storage** consists of 4,096 such bins $b^0, b^1, \ldots, b^{4095}$ connected in parallel as indicated in Fig. 5.

When the machine is idle all components of the trunk are in state 0, and all cells of the bin can be cleared to state 0 manually by activating the key at the upper left corner of Fig. 4. That key's activation at such a time τ

prevents any signal that may be at $\underline{b}_i^n(\tau)$ from reaching $\underline{h}_i^n$, whence $\underline{h}_i^n(\tau) = 0$ and $\underline{b}_i^n(\tau + 1) = 0$ regardless of the value of $\underline{b}_i^n(\tau)$. Having thus cleared all cells to 0, a 1 may be inserted in any cell by activating the key at that cell's right. For if all control wires are in state 0 and the key at the left is not activated, then activating the right-hand key at time τ will cause a signal to pass through the conjunction element below $\underline{h}_i^n$ up to make $\underline{h}_i^n(\tau) = 1$, whence $\underline{b}_i^n(\tau + 1) = 1$, and the cell will continue to store the 1 until caused to change by the activation of the left-hand key or of some control wire. Thus the keys can be used to load each cell with its initial bit of information, and the lights provide a crude way of getting information out of the bin, since each one is visibly on or off according as its cell contains a 1 or a 0. We assume that the keys are inactive in the following discussion.

Three control wires lead to each bin, a distinct set to every bin. Fig. 4 shows the _n_th bin and its control wires $\underline{R}_p^n$, $\underline{R}_t^n$, and $\underline{T}^n$. Its _i_th cell $\underline{b}_i^n$ is connected to the _i_th component $\underline{t}_i$ of the trunk.

The control wire $\underline{T}^n$ is the transmit wire for bin $\underline{n}$. When it is activated, the bit in each cell enters the appropriate component of the trunk, passing from $\underline{b}_i^n$ through the lower conjunction element onto t_i, whence $t_i \equiv T^n \circ b_i^n$.

It is convenient to be able to alter an instruction by changing its address without modifying the operand. Hence the wiring of the first four cells of each bin is different from that of the last twelve cells, as shown in Fig. 4. The control wire R_p^n is the *partial reception* control for bin n. It connects, by way of the disjunction element, to the last twelve cells b_4^n, b_5^n, ..., b_{15}^n. If it is activated at time τ (by instruction 11 of Fig. 2), and R_t^n is not activated, then whatever bit is on trunk component t_i ($i = 4, 5, \ldots, 15$) at time τ will flow through the upper left conjunction element of cell b_i^n onto h_i, from which it will emerge at $\tau + 1$ to occupy b_i^n, in which case $b_i^n(\tau+1) \equiv t_i(\tau)$ for $i = 4, 5, \ldots, 15$, and $b_i^n(\tau+1) \equiv b_i^n(\tau)$ for $i = 0, 1, 2, 3$.

The control wire R_t^n is the *total reception* control for bin n. It connects directly to the first four cells and by way of the disjunction element to the last twelve cells also. If it is activated at time τ (by instruction 10), then whatever bit is on trunk component t_i at time τ will occupy cell b_i^n at time $\tau + 1$; in this case $b_i^n(\tau + 1) \equiv t_i(\tau)$.

The storage bin's behavior is described more compendiously by the equations

$$b_i^n(\tau+1) \equiv \left[R_t^n(\tau) \circ t_i(\tau) \vee \overline{R}_t^n(\tau) \circ b_i^n(\tau)\right] \quad \text{for } i = 0, \ldots, 3, \text{ and}$$

$$b_i^n(\tau+1) \equiv \left\{\left[R_t^n(\tau) \vee R_p^n(\tau)\right] t_i(\tau) \vee \overline{R}_t^n(\tau) \circ \overline{R}_p^n(\tau) \circ b_i^n(\tau)\right\} \quad \text{for } i = 4, \ldots, 15.$$

4.3. Address Decoder.

For the computer to use the information in its parallel storage, it must have some method of switching to or selecting a specified location there. Twelve binary digits suffice to specify any storage bin uniquely. The Address Decoder is a switching mechanism which activates the proper storage bin when it receives the twelve digit number which is the address of that bin.

The Address Decoder has 4,096 distinct output wires $\underline{p}_0$, $\underline{p}_1$, ..., $\underline{p}_{4095}$, each connected to a distinct storage bin of the parallel storage. Each distinct $\underline{p}_i$ is activated by a different set of twelve digits which may be carried by the Address Decoder's twelve input wires $\underline{s}_4$, $\underline{s}_5$, ..., $\underline{s}_{15}$. The functional connection is accomplished through the use of 4,096 threshold-12 elements, as shown in Fig. 5.[5] The same twelve input wires are connected to each of the 4,096 threshold-12 elements, those connections being described by the equations: $\underline{p}_0 \equiv (\bar{\underline{s}}_4\bar{\underline{s}}_5\cdots\bar{\underline{s}}_{14}\bar{\underline{s}}_{15})$, $\underline{p}_1 \equiv (\bar{\underline{s}}_4\bar{\underline{s}}_5\cdots\bar{\underline{s}}_{14}\underline{s}_{15})$, ..., $\underline{p}_{4094} \equiv (\bar{\underline{s}}_4\underline{s}_5\cdots\underline{s}_{14}\underline{s}_{15})$, $\underline{p}_{4095} \equiv (\underline{s}_4\underline{s}_5\cdots\underline{s}_{14}\underline{s}_{15})$.

General directions for the parallel storage are carried by control wires $\underline{S}_{rp}$, $\underline{S}_{rt}$, and $\underline{S}_t$ (Figs. 1 and 5), at most one of which is in state 1 at any time. Each of these control wires is connected to 4,096 different conjunction elements, whose other input wires are the 4,096 distinct $\underline{p}_i$'s. Since at most one of the control wires $\underline{S}_{rp}$, $\underline{S}_{rt}$, $\underline{S}_t$ is activated at any time, and exactly one of the Address Decoder's output wires $\underline{p}_0, \underline{p}_1, \ldots, \underline{p}_{4095}$ is activated at any given time, there is at any given time at most one bin $\underline{n}$ for which any of the control wires $\underline{R}_p^n$, $\underline{R}_t^n$, $\underline{T}^n$ is activated at that time, and, if there is such a bin $\underline{n}$, at most one of those three control wires $\underline{R}_p^n$, $\underline{R}_t^n$, $\underline{T}^n$ is activated at that time. The equations which describe the circuits diagrammed in Fig. 5 are of the form: $\underline{R}_p^n \equiv (\underline{S}_{rp} \cdot \underline{p}_n)$, $\underline{R}_t^n \equiv (\underline{S}_{rt} \cdot \underline{p}_n)$, $\underline{T}^n \equiv (\underline{S}_t \cdot \underline{p}_n)$.

For example, in the execution of instruction 4 of Fig. 2, ADD X, the operand 0100 of that instruction causes control wire $\underline{S}_t$ to be stimulated and the address $\underline{x}_4\underline{x}_5\ldots\underline{x}_{15}$ enters the input wires of the Address Decoder to stimulate its output wire $\underline{p}_x$. That stimulates $\underline{T}^x$, the output wire of a conjunction element whose two input wires are $\underline{S}_t$ and $\underline{p}_x$, to make the number in bin $\underline{X}$ pass onto the trunk where it is available to the ARITHMETIC UNIT.

Here and throughout this paper we speak as if the electrical response of a circuit were instantaneous, which is not strictly accurate. For our remarks to be made unobjectionable we need only make the duration of the time interval we refer to as a "moment" sufficiently large to permit the signal to pass from any part of the machine to any other part during that interval. Our "moments" need not all be of the same duration, of course, and intervals of any different durations may lie between what we refer to as "successive moments." It should be kept in mind that the shorter the durations of these "moments" and the shorter the time lapses between them, the higher the speed of computation that can be achieved by the machine.

These remarks should suffice to indicate that the states 0 and 1 of our wires need not be pulses, but can alternatively be static states. It is, of course, part of the idealized representational function of our diagrams that they permit of either interpretation.

In practice a relatively uncomplicated physical device may correspond to a very complex logical net; no one-to-one correspondence is suggested between the elements of our diagrams and the physical components of the computing machine which realizes them. For example, our general design for the parallel storage (less the keys) can be realized physically without using separate delay lines, as by a Williams' tube cathode ray storage device which consists of many tubes and associated circuits. The delay elements in our diagrams represent the fact that information is stored, but do not represent the specific physical devices that may be used to accomplish that storage function. Our design is intended to represent the logical role rather than the physical constitution of a computer and its parts. Hence many of the details of the circuits involved in the Williams' tube storage (pulse shaper, etc.) are not represented by our diagrams. Nor is the breakdown into parts similar, for a single tube will store bits from many bins (say, 1024 of them), with the contents of a single bin distributed over sixteen different tubes.

4.4. Serial Storage.

In addition to the relatively limited parallel storage already described, our computer has a theoretically infinite serial storage,[6] which supplements the parallel storage. The *serial storage* consists of sixteen parallel strings of cells, every cross section of which may be regarded as a storage bin. Our computer is so designed that exactly one storage bin belongs to both the parallel storage and the serial storage.[7] For definiteness, we specify it to be bin 4095 of the parallel storage, which must be given somewhat more complicated cells to enable it to play its dual role. In Fig. 6, which shows only three cells of the topmost string, the middle cell diagrammed belongs to this special bin. Each string is to be thought of as extending indefinitely in both directions.

It is clear from the diagram that if neither of the control wires $\underline{S}_b, \underline{S}_f$ is activated, every bin of the serial storage save $\underline{b}^{4095}$ continues to store whatever information it contains, and if $\underline{R}_p^{4095} \equiv \underline{R}_t^{4095} = 0$, then $\underline{b}^{4095}$ also continues to store whatever information it contains. Now, if control wire $\underline{S}_b$ ($\underline{S}_f$) is activated at time τ, the internal circulation or storage of information in each cell is interrupted at the threshold-3 element in each ordinary cell and at the threshold-4 element in each special cell of the shared bin, and any bit of information in each cell will pass up the wire to the right of its delay element, through the left (right) conjunction element above, and down to the input wire of the delay element of the cell to its left (right).

If we assign the labels ..., $\beta_i^{-2}, \beta_i^{-1}, \beta_i^{0}, \beta_i^{1}, \beta_i^{2}$, ... to the cells of the serial storage, where β_i^0 is $\underline{b}_i^{4095}$, the behavior of the serial storage is described by the expression

$$\bar{\underline{R}}_p^{4095} \cdot \bar{\underline{R}}_t^{4095} \supset \left\{ \beta_i^n(\tau+1) \equiv \left[\underline{S}_b(\tau) \cdot \beta_i^{n+1}(\tau) \vee \underline{S}_f(\tau) \cdot \beta_i^{n-1}(\tau) \vee \bar{\underline{S}}_b(\tau) \cdot \bar{\underline{S}}_f(\tau) \cdot \beta_i^n(\tau) \right] \right\},$$

where $\underline{n} = \ldots, -2, -1, 0, 1, 2, \ldots$.

If a word located in an ordinary bin of the serial storage is to be used, the contents of the serial storage must be shifted backward or forward until that word occupies the special bin $\underline{b}^{4095}$, from which it is immediately available to the computer as described in Section 4.2. Hence the greater capacity of the serial storage is purchased, so to speak, at the expense of ready availability of its contents.

At this point the analogy should be noted again between our diagrammatic conventions and the process of definition. Just as a single term can be introduced as an abbreviation for a whole sequence of other terms, so some parts of our diagrams are to be understood as definitional abbreviations for other, more complicated parts. We have already pointed out that each right-hand diagram in Figs. 3b, c, d, f, and g is a definitional abbreviation of the diagram to its left. Similarly, in Fig. 5 each box $\underline{b}^n$ is to be understood as a shorthand notation for the more detailed diagram of storage bin $\underline{b}^n$ presented in Fig. 4. (Both, it should be noted, are connected to the same three control wires and the same sixteen components of the trunk.) And the block labeled STORAGE in Fig. 1 is a definitional abbreviation of the complete net whose parts are represented in Figs. 5 and 6, that is, of the combined and interconnected parallel and serial storages.

5. ARITHMETIC UNIT.

5.1. Machine Arithmetic.

Before describing the operation of the ARITHMETIC UNIT we must discuss the arithmetic of the machine.

A binary numerical expression $+0.x_1 \ldots x_{15}$ (≥ 0) is directly represented in the machine by the word $0x_1 \ldots x_{15}$, and a binary number $-0.x_1 \ldots x_{15}$ (< 0) by the word $2-.x_1 \ldots x_{15}$. For example, $+.110\ldots0$ ($+3/4$) is directly represented by $0.110\ldots0$, and $-.110\ldots0$ ($-3/4$) by $1.010\ldots0$. Clearly any binary number x in the range $-1 \leq x < 1$ can be directly represented in the machine to within 2^{-15}. Numbers outside this range can be indirectly represented in the machine through shifting them into the range by multiplying them by 2^{-n} with some appropriate n.

The binary point of a number is not directly represented in the machine. However, it is convenient to imagine, and sometimes to show, a binary point to the right of the leftmost bit of every machine word that directly represents a number.

It will be convenient to introduce an additional convention at this time. Just as a symbol like b_i may either name a wire or symbolize a function (which takes on one of the two values 0,1 at each time τ) (see Section 3), so a symbol like b may either name a sequence of sixteen wires $\underline{b}_0, \ldots, \underline{b}_{15}$ or symbolize a function (which takes on one of the 2^{16} values: 0.0...00, 0.0...01, ..., 1.1...10, 1.1...11 at each time τ). Our earlier use of $\underline{t}$ (Section 2) to stand for the trunk consisting of the wires $\underline{t}_0, \ldots, \underline{t}_{15}$ accords with this convention.

We shall introduce four operations on machine words: machine addition, machine subtraction, machine doubling, and machine halving; defining them so as to maintain a certain correspondence with the analogous operations of ordinary arithmetic. Where $\underline{x}$ and $\underline{y}$ are machine words which directly represent the numbers α and β, respectively, and $\underline{z}$ is the result of performing machine addition on $\underline{x}$ and $\underline{y}$, then if $\alpha + \beta$ can be directly represented in the machine, it is directly represented by $\underline{z}$, whereas if $\alpha + \beta$ can not be directly represented in the machine, $\underline{z}$ directly represents a number congruent to $\alpha + \beta$ modulo 2. Similar remarks apply for the other three operations also. We will hereafter use the signs $\underline{x}+\underline{y}$, $-\underline{x}+\underline{y}$, $2\underline{y}$, and $(1/2)\underline{y}$ to denote our four machine operations. Basically, the machine does arithmetic modulo 2 with a precision of one part in 2^{15}. For example, 0.110...0 + 0.110...0 = 1.100...0, i.e., $3/4 + 3/4 = -(1/2)$, since $3/4 + 3/4 \equiv -(1/2)$ modulo 2; and halving 0.0...01 (2^{-15}) gives 0, while halving 1.1...11 (-2^{-15}) gives -2^{-15}.

The machine operations of addition, subtraction, doubling, and halving are the only machine arithmetic operations for which there are instructions in Fig. 2. However, other machine operations (e.g., multiplication, division), as well as machine operations on numbers which cannot be directly represented in the machine, can be programmed; that is, there exist routines (sequences of instructions of Fig. 2) for making the machine perform these additional operations.

Our machine arithmetic operations are to be performed by nets constructed out of our logical elements, and these nets must all be realized by our ARITHMETIC UNIT. The machine arithmetic operations performed on sixteen digit words ($\underline{x}$,$\underline{y}$) must be defined in terms of logical operations performed on the several bits ($\underline{x}_i$, $\underline{y}_i$) of those words.

We first define machine addition. Our task is to express

(A) $\underline{z} = \underline{x} + \underline{y}$

as a logical relation between $\underline{x}_i$, $\underline{y}_i$, and $\underline{z}_i$. It is helpful to define the auxiliary word ξ whose $\underline{i}$-1st bit ξ_{i-1} ($\underline{i} > 0$) is the carry digit from the addition of the three summands: $\underline{x}_i$, $\underline{y}_i$, and the carry digit ξ_i. It is clear that ξ_{15} is 0, and that $\underline{z}_i$ is 1 if and only if an odd number of the summands $\underline{x}_i$, $\underline{y}_i$, ξ_i are 1. The result concerning inequivalence established in Section 3 enables us to express this rule by

(A_1) $\quad \underline{z}_i \equiv (\underline{x}_i \not\equiv \underline{y}_i \not\equiv \xi_i)$.

And since ξ_{i-1} is 1 just in case at least two of the three summands are 1, we have the equations:

(A_2) $\quad \xi_{i-1} \equiv (\underline{x}_i\underline{y}_i \vee \underline{x}_i\xi_i \vee \underline{y}_i\xi_i)$ for $\underline{i} > 0$,

(A_3) $\quad \xi_{15} = 0$.

Before defining machine subtraction we introduce the machine operation of *complementation* $\underline{C}$, whose field consists of words. We define $\underline{C}(\underline{x})$ to be the machine representation of that number within the range $-1 \leq \underline{x} < 1$ which is congruent modulo 2 to $2-\underline{x}$. The complementation operation is very useful because a negative number $-.\underline{x}_1\ldots\underline{x}_{15}$ is directly represented in our machine by $\underline{C}(0.\underline{x}_1\ldots\underline{x}_{15})$. Machine complementation is related to logical complementation (negation) in the following way. We define

$$\overline{\underline{x}} =_{df} \overline{\underline{x}}_0.\overline{\underline{x}}_1\ldots\overline{\underline{x}}_{15};$$

and refer to $\overline{\underline{x}}$ as the *bitwise complement* of $\underline{x}$. Now 2 is the arithmetic sum of 1.11...11 and 2^{-15}, and clearly $\overline{\underline{x}} = 1.11\ldots11-\underline{x}$, whence

$$\underline{C}(\underline{x}) = \overline{\underline{x}}_0.\overline{\underline{x}}_1\ldots\overline{\underline{x}}_{15} + 0.0\ldots01.$$

Machine complementation is therefore performed on a number by adding 2^{-15} to the bitwise complement of that number.

Since the arithmetic operation of subtracting α from β is equivalent to the addition of the negative of α to β, we can define machine subtraction in terms of complementation and addition. Thus, the formula for **machine subtraction**

(S) $\quad z = -x + y$

becomes

$$z = C(x) + y = \bar{x} + y + 0.0\ldots01.$$

By using (A_1) and (A_2) we can translate $z = \bar{x} + y$ into logical terms; and since ξ_{15} is one of the summands in the rightmost digital position, the addition of 2^{-15} can be accomplished by setting ξ_{15} equal to 1. Thus, machine subtraction of the word x from the word y can be defined in terms of the following logical relations among the bits x_i, y_i, ξ_i, and z_i.

(S_1) $\quad z_i \equiv (\bar{x}_i \not\equiv y_i \not\equiv \xi_i)$,

(S_2) $\quad \xi_{i-1} \equiv (\bar{x}_i y_i \vee \bar{x}_i \xi_i \vee y_i \xi_i)$ for $i > 0$,

(S_3) $\quad \xi_{15} = 1$.

To define **machine doubling** we want to express

(D) $\quad z = 2y$

in logical terms. Since a 1 in the ith position has twice the value of a 1 in the $i+1$st position, doubling consists merely of shifting each digit of y left one binary position with respect to the binary point. We express (D) in terms of the logical operations on the bits y_i by

(D_1) $\quad z_i \equiv y_{i+1}$ for $i < 15$,

(D_2) $\quad z_{15} = 0$.

Similarly, halving is accomplished by a right shift. We define machine halving,

(H) $\underline{z} = (1/2)\underline{y}$,

by

(H_1) $\underline{z}_{\underline{i}} \equiv \underline{y}_{\underline{i}-1}$ for $\underline{i} > 0$,

(H_2) $\underline{z}_0 \equiv \underline{y}_0$.

5.2. Operation of the CONTROL.

Instructions 4, 5, 6, 7, 8, 9, 10, and 11 of Fig. 2 all involve the ARITHMETIC UNIT. Before describing how the ARITHMETIC UNIT is to function in their execution, we must explain briefly how the CONTROL affects the ARITHMETIC UNIT. Exactly how the CONTROL performs the functions we are going to describe will be explained later in Section 6.

Since the instructions involving the ARITHMETIC UNIT all refer to a "number in the ARITHMETIC UNIT", it will be convenient to have a way of referring to this number. We will stipulate that it is located on the sequence of wires $\underline{a}$.

At any time τ the wire sequence $\underline{d}_0$, $\underline{d}_1$, $\underline{d}_2$, $\underline{d}_3$ is in exactly one of sixteen distinct states. These sixteen states, together with the state of the CONTROL CLOCK, completely determine the states of the thirteen control wires labeled $\underline{A}$, $\underline{S}$, and $\underline{C}$ with appropriate subscripts. (When the machine is inoperative, none of the wires is stimulated.) Fig. 2 tells what control wires are activated for the various states of $\underline{d}_0$, $\underline{d}_1$, $\underline{d}_2$, $\underline{d}_3$ when the machine is operating; control wires not marked as active are stipulated to be inactive. The state 1111 can occur only by a mistake in programming; when it does occur, no control wires are activated, and it produces no effect.

When one of the instructions 4, 5, 6, 7 is being executed, the CONTROL will activate $\underline{S}_t$ and also send the address $\underline{X}$ to $\underline{s}_4, \ldots, \underline{s}_{15}$, thereby causing the STORAGE to transmit the number in storage bin $\underline{X}$ to the trunk $\underline{t}$. Hence we may assume that when these instructions are being executed the number in storage bin $\underline{X}$ is on $\underline{t}$ and therefore available to the ARITHMETIC UNIT.

When instruction 10 or 11 is being executed, the CONTROL will activate $\underline{S}_{rt}$ or $\underline{S}_{rp}$, respectively, and will also send the address $\underline{X}$ to $\underline{s}_4, \ldots, \underline{s}_{15}$, causing bin $\underline{X}$ to receive in the desired fashion any number on the trunk $\underline{t}$. Hence we must so design the ARITHMETIC UNIT that these instructions cause the contents of $\underline{a}$ to occupy $\underline{t}$.

We may summarize the effect of the CONTROL on the ARITHMETIC UNIT as follows. When one of the instructions 4 through 11 is being executed, control wires $\underline{A}_r$, $\underline{A}_h$, $\underline{A}_c$, $\underline{A}_{ls}$, $\underline{A}_{rs}$, $\underline{A}_{tt}$ are in the states shown in Fig. 2; at any other time these wires are all inactive. When instruction 4, 5, 6, or 7 is being executed, the number in bin $\underline{X}$ is also on the trunk $\underline{t}$. Finally, when instruction 10 or 11 is being executed, the CONTROL causes whatever number the ARITHMETIC UNIT places on $\underline{t}$ to be properly received by the correct bin.

5.3. The Functions of the ARITHMETIC UNIT.

In the previous subsection we described every possible way for information to be directed to the ARITHMETIC UNIT by the CONTROL. In this subsection we consider three general functions performed by the ARITHMETIC UNIT on the basis of this information, and explain how it performs the first two.

The first function is the transmission of information. When either instruction 10 or 11 is executed, the ARITHMETIC UNIT is to transmit its contents onto the trunk $\underline{t}$. If either of these instructions is executed (see Fig. 2) $\underline{A}_{tt}$ will be stimulated, hence we want the machine to realize the expression

$$\underline{A}_{tt} \supset (\underline{a}_i \equiv \underline{t}_i).$$

This function is accomplished by the conjunction elements to the right in Fig. 7. The ARITHMETIC UNIT is also to supply the bits $\underline{a}_0$ and $\underline{a}_{15}$ to the CONTROL, which is accomplished simply by running wires from these cells to the CONTROL as indicated in Fig. 7. We will simplify the discussion of the remainder of this section by ignoring the transmission function of the ARITHMETIC UNIT (and the relevant parts of Fig. 7).

The second function performed by the ARITHMETIC UNIT is storage. It is clearly required that when the ARITHMETIC UNIT is not executing any of the instructions 4 through 9, it must continue to store the number $\underline{a}$. Inspection of Fig. 2 shows that $\underline{A}_r$ is activated when and only when one of the instructions 4 through 9 is being executed. Hence when $\underline{A}_r(\tau) = 0$ we want $\underline{a}(\tau) = \underline{a}(\tau+1)$. This storage may be accomplished by using sixteen delay elements, with input wires $\underline{r}_0, \ldots, \underline{r}_{15}$ and output wires $\underline{a}_0, \ldots, \underline{a}_{15}$, as in Fig. 7, so connected that

$$\overline{\underline{A}}_r \supset (\underline{r}_i \equiv \underline{a}_i).$$

Since $\underline{a}_i(\tau+1) = \underline{r}_i(\tau)$, the ARITHMETIC UNIT does perform the desired storage function.

The third function performed by the ARITHMETIC UNIT is the modification of its contents from $\underline{a}(\tau)$ to $\underline{a}(\tau+1)$ in accordance with instruction 4, 5, 6, 7, 8, or 9, which we shall refer to as *arithmetic instructions*. The design of a net which will realize the arithmetic instructions is the most complicated and difficult part of the task of designing the machine. To it we devote both the following subsection and Appendix E.

5.4. Execution of the Arithmetic Instructions.

In this subsection we shall derive a set of formulas which the ARITHMETIC UNIT must realize for the arithmetic instructions to be executed. We derive them by combining the various arithmetic instructions with the equations (A), (S), (D), and (H) of Section 5.1, which define the various machine arithmetic operations. We note that the result of an operation performed at time $\mathcal{T}$ is to appear at $\underline{r}$ at time $\mathcal{T}$ and at $\underline{a}$ at time $\mathcal{T}+1$.

Instruction 4 demands that $\underline{r}$ contain the result of performing the machine addition of the number at $\underline{t}$ to the number at $\underline{a}$. For the ARITHMETIC UNIT to fulfill its intended function, it must realize the conditional

(A^m) If ADD X is enjoined, then $\underline{r} = \underline{t} + \underline{a}$.

Now by Fig. 2, ADD X is the only instruction which involves the stimulation of $\underline{A}_r$ and $\underline{A}_h$ but not $\underline{A}_c$. Hence the antecedent of (A^m) may be replaced by $\underline{A}_r\underline{A}_h\overline{\underline{A}}_c$. And a translation of the consequent of (A^m) into a logical formula concerning the states of individual wires may be made by using (A_1), (A_2), and (A_3), substituting $\underline{r}_i$ for $\underline{z}_i$, $\underline{t}_i$ for $\underline{x}_i$, and $\underline{a}_i$ for $\underline{y}_i$. In the ARITHMETIC UNIT we are constructing, the auxiliary word ξ will be realized by the set of wires $\underline{c}_0, \underline{c}_1, \ldots, \underline{c}_{15}$; hence we also substitute $\underline{c}_i$ for ξ_i in the (A) equations. We thus obtain the following formulas for the ARITHMETIC UNIT to realize:

(A^m_1) $\underline{A}_r\underline{A}_h\overline{\underline{A}}_c \supset \{\underline{r}_i \equiv (\underline{t}_i \not\equiv \underline{a}_i \not\equiv \underline{c}_i)\}$,

(A^m_2) $\underline{A}_r\underline{A}_h\overline{\underline{A}}_c \supset \{\underline{c}_{i-1} \equiv (\underline{t}_i\underline{a}_i \vee \underline{t}_i\underline{c}_i \vee \underline{a}_i\underline{c}_i)\}$ for $\underline{i} > 0$,

(A^m_3) $\underline{A}_r\underline{A}_h\overline{\underline{A}}_c \supset \{\underline{c}_{15} = 0\}$.

A set of formulas for the ARITHMETIC UNIT to realize to execute instruction 5 may be derived from the (S) equations in a parallel way. We thus obtain

(S^m) If SUBTRACT X is enjoined, then $\underline{r} = -\underline{t} + \underline{a}$,

(S^m_1) $\underline{A}_r\underline{A}_h\underline{A}_c \supset \{\underline{r}_i \equiv (\overline{\underline{t}}_i \not\equiv \underline{a}_i \not\equiv \underline{c}_i)\}$,

(S^m_2) $\underline{A}_r\underline{A}_h\underline{A}_c \supset \{\underline{c}_{i-1} \equiv (\overline{\underline{t}}_i\underline{a}_i \vee \overline{\underline{t}}_i\underline{c}_i \vee \underline{a}_i\underline{c}_i)\}$ for $\underline{i} > 0$,

(S^m_3) $\underline{A}_r\underline{A}_h\underline{A}_c \supset \{\underline{c}_{15} = 1\}$.

Instruction 6 demands the replacement of $\underline{r}$ by $\underline{t}$. For it to be executed, the ARITHMETIC UNIT must realize the conditional

(T^m) If TRANSFER X is enjoined, then $\underline{r} = \underline{t}$.

Consulting Fig. 2, we find that this can be represented as

(T^m_1) $\underline{A}_r\overline{\underline{A}}_h\overline{\underline{A}}_c \supset (\underline{r}_i \equiv \underline{t}_i)$.

Instruction 7 demands that $\underline{r}$ be replaced by $-\underline{t}$; for it to be executed the ARITHMETIC UNIT must realize the conditional

(TC^m) If TRANSFER COMPLEMENT X is enjoined, then $\underline{r} = \underline{C}(\underline{t})$.

Since $-\underline{t} = -\underline{t} + 0$, the formulas required here can be regarded as special cases of the (S) formulas, with 0 substituted for $\underline{y}$. After that substitution has been made, the resulting formulas easily simplify to

(TC_1^m) $\quad \underline{A}_r\overline{\underline{A}}_h\underline{A}_c \supset \{\underline{r}_i \equiv (\overline{\underline{t}}_i \not\equiv \underline{c}_i)\}$,

(TC_2^m) $\quad \underline{A}_r\overline{\underline{A}}_h\underline{A}_c \supset \{\underline{c}_{i-1} \equiv (\overline{\underline{t}}_i\underline{c}_i)\}$ for $\underline{i} > 0$,

(TC_3^m) $\quad \underline{A}_r\overline{\underline{A}}_h\underline{A}_c \supset \{\underline{c}_{15} = 1\}$,

which the ARITHMETIC UNIT must realize to execute instruction 7.

Instruction 8 demands that $\underline{r}$ be replaced by $2\underline{a}$; for it to be executed the ARITHMETIC UNIT must realize the conditional

(D^m) $\quad$ If DOUBLE is enjoined, then $\underline{r} = 2\underline{a}$.

Again substituting $\underline{r}$ for $\underline{z}$ and $\underline{a}$ for $\underline{y}$, this time in the (D) formulas, and consulting Fig. 2, we obtain

(D_1^m) $\quad \underline{A}_r\underline{A}_{ls} \supset \{\underline{r}_i \equiv \underline{a}_{i+1}\}$ for $\underline{i} > 0$,

(D_2^m) $\quad \underline{A}_r\underline{A}_{ls} \supset \{\underline{r}_{15} = 0\}$.

These formulas must be realized by the ARITHMETIC UNIT for it to execute instruction 8.

Instruction 9 demands that $\underline{r}$ be replaced by $(1/2)\underline{a}$; for it to be executed the ARITHMETIC UNIT must realize the conditional

(H^m) $\quad$ If HALVE is enjoined, then $\underline{r} = (1/2)\underline{a}$.

Again we substitute $\underline{r}$ for $\underline{z}$ and $\underline{a}$ for $\underline{y}$, this time in the (H) formulas, and consult Fig. 2, to obtain

(H_1^m) $\quad \underline{A}_r\underline{A}_{rs} \supset \{\underline{r}_i \equiv \underline{a}_{i-1}\}$ for $\underline{i} > 0$,

(H_2^m) $\quad \underline{A}_r\underline{A}_{rs} \supset \{\underline{r}_0 \equiv \underline{a}_0\}$.

These must be realized by the ARITHMETIC UNIT if it is to execute instruction 9.

We have now completed the task of deriving a set of formulas which the ARITHMETIC UNIT must realize if it is to do its part in executing the arithmetic instructions. These are the fourteen formulas whose labels have superscript $\underline{m}$ and subscripts 1, 2, ~~or~~ 3. These fourteen formulas, together with the storage formulas (see Section 5.3)

(K) $\quad \overline{\underline{A}}_r \supset (\underline{r}_i \equiv \underline{a}_i)$

and

(K') $\quad \underline{a}_i(\mathcal{T}+1) \equiv \underline{r}_i(\mathcal{T})$,

provide a complete specification of the ARITHMETIC UNIT (since we have agreed to neglect its transmission function). Any logical net which realizes these sixteen *defining formulas* will be a satisfactory ARITHMETIC UNIT. It is readily verified that Fig. 7 is such a net, by observing that the last formula above is satisfied by the delay elements, and then taking each of the remaining formulas in turn and determining that under the conditions stated each $\underline{r}_i$ has the proper relations to $\underline{a}_i$ and $\underline{t}_i$. Performing this verification will facilitate understanding how the ARITHMETIC UNIT works.

6. CONTROL.

6.1. ADDRESS COUNTER.

Apart from "jump" instructions (12, 13, 14 of Fig. 2) the computer executes in order the instructions stored in a sequence of bins of the parallel storage. To do so it must count off those instructions as they are executed, and this function is performed by the ADDRESS COUNTER diagrammed in Fig. 8.

Each of its twelve cells $\underline{a}'_4$, $\underline{a}'_5$, ..., $\underline{a}'_{15}$ can receive and store one bit of information, so the ADDRESS COUNTER as a whole can receive and store a twelve digit number $\underline{x}_4\underline{x}_5\ldots\underline{x}_{15}$. If all control wires are in state 0 and no keys are activated, each cell $\underline{a}'_i$ continues to store whatever information it contains. Any number in the ADDRESS COUNTER is the address of some bin of the parallel storage. To load the ADDRESS COUNTER with the address of the bin containing the first instruction we want the machine to execute, we must be able to change the contents of the ADDRESS COUNTER. The keys shown in Fig. 8 enable us to make any desired change. Activating the key at the lower left (when the machine is idle) clears all of its cells to 0, and activating the key of cell $\underline{a}'_i$ at time $\mathcal{T}$ makes $\underline{a}'_i(\mathcal{T}+1) = 1$; thus we can load the ADDRESS COUNTER with any number we please.

As remarked at the bottom of the TABLE OF INSTRUCTIONS on Fig. 2, at every even numbered time $\tau = 2\underline{k}$ when the machine is operating, control wires $\underline{S}_t$ and $\underline{C}_t$ are activated. At any such time, then, $\underline{C}_t = 1$ permits the bit stored in cell $\underline{a}'_i$ to pass through the rightmost conjunction element onto $\underline{s}_i$ ($\underline{i} = 4, 5, \ldots, 15$). Since $\underline{s}_4, \underline{s}_5, \ldots, \underline{s}_{15}$ are input wires to the Address Decoder (see Fig. 5), and since $\underline{S}_t$ is also activated then, that storage bin whose address is contained in the ADDRESS COUNTER at any even numbered time $\tau = 2\underline{k}$ will transmit its contents onto the trunk at that time.

Hence when the computer begins to solve a problem the first instruction it executes is the one stored in that bin whose address is stored in the ADDRESS COUNTER, usually bin 0. To keep track, whenever the ADDRESS COUNTER sends an instruction from the $\underline{n}$th bin onto the trunk it must count, that is, add a 1 to the number it contains at that time.

(Any twelve digit number $\underline{x}_4\underline{x}_5\ldots\underline{x}_{15}$ contained in the ADDRESS COUNTER is the address of some bin of the parallel storage, and can also constitute the address part of an instruction word, as in $\underline{d}_0\underline{d}_1\underline{d}_2\underline{d}_3\underline{x}_4\underline{x}_5\ldots\underline{x}_{15}$. Whenever we consider the numerical aspect of a word we must keep in mind the discussion of range presented in Section 5, and the convention that a binary point is imagined to follow the leftmost bit of every number word. This convention should apply to instruction words too, for it is often convenient to perform arithmetic operations on instruction words, as will be illustrated in Appendix D. From this point of view the 4096 addresses of the 4096 bins of the parallel storage are 0.000000000000000, 0.000000000000001, ..., 0.000111111111111, and the number in the ADDRESS COUNTER is $0.000\underline{x}_4\underline{x}_5\ldots\underline{x}_{15}$ or $(\underline{x}_4\underline{x}_5\ldots\underline{x}_{15})2^{-15}$. Consequently the address of the next bin after that one is $(\underline{x}_4\underline{x}_5\ldots\underline{x}_{15})2^{-15}+2^{-15}$. It will, however, be convenient to continue to speak in this connection of the number in the ADDRESS COUNTER as $\underline{x}_4\underline{x}_5\ldots\underline{x}_{15}$, and of its successor as that number $+ 1$.)

Let us suppose that the number in the ADDRESS COUNTER at time τ is $\underline{x}_4\underline{x}_5\ldots\underline{x}_{15}$, and to it we wish to add 1, which is $\underline{y}_4\underline{y}_5\ldots\underline{y}_{15}$ where $\underline{y}_{15} = 1$ and $\underline{y}_4 = \underline{y}_5 = \ldots = \underline{y}_{14} = 0$. The sum $\underline{z}_4\underline{z}_5\ldots\underline{z}_{15}$ of these two numbers is recursively defined (see Section 5) by the equations:

$$\underline{z}_i \equiv (\underline{x}_i \not\equiv \underline{y}_i \not\equiv \xi_i)$$

and

$$\xi_{i-1} \equiv (\underline{x}_i\underline{y}_i \vee \underline{x}_i\,\xi_i \vee \underline{y}_i\,\xi_i) \text{ where } \xi_{15} = 0.$$

Since $\xi_{15} = 0$ and $\underline{y}_{15} = 1$, by the first defining equation we have $\underline{z}_{15} \equiv \overline{\underline{x}}_{15}$, and by the second defining equation we have $\xi_{14} \equiv \underline{x}_{15}$. Now for every $\underline{i} < 15$, $\underline{y}_i = 0$, whence $\xi_{i-1} \equiv \underline{x}_i\,\xi_i$, and also $\underline{z}_i \equiv (\underline{x}_i \not\equiv \xi_i)$. These formulas must be realized by the ADDRESS COUNTER if it is to perform its counting function correctly.

For the formulas developed in the preceding paragraph to be realized by a circuit, that circuit must contain wires corresponding to $\underline{x}_4, \underline{x}_5, \ldots, \underline{x}_{15}$, to $\xi_4, \xi_5, \ldots, \xi_{14}$, and to $\underline{z}_4, \underline{z}_5, \ldots, \underline{z}_{15}$, and these wires must be so connected that their behavior is described by those formulas. If net wires $\underline{a}'_i$, $\underline{c}'_i$, and $\underline{r}'_i$ correspond to $\underline{x}_i$, ξ_i, and $\underline{z}_i$, respectively, those wires must realize the equations $\underline{r}'_i \equiv (\underline{a}'_i \not\equiv \underline{c}'_i)$ and $\underline{c}'_{i-1} \equiv \underline{a}'_i\underline{c}'_i$ for $\underline{i} < 15$, $\underline{r}'_{15} \equiv \bar{\underline{a}}'_{15}$, and $\underline{c}'_{14} \equiv \underline{a}'_{15}$.

The diagram in Fig. 8 satisfies the preceding equations when $\underline{C}_r = 0$ and $\underline{C}_t = 1$. By assumption, $\underline{a}' \equiv \underline{x}$, and since $\underline{C}_r = 0$ and $\underline{C}_t = 1$ at every even numbered moment $\tau = 2\underline{k}$, at every such moment $\underline{r}'$ will contain the immediate successor of the number in $\underline{a}$, and that successor will occupy $\underline{a}$ at the following moment $\tau = 2\underline{k}+1$.

If at an odd numbered moment $\tau = 2\underline{k}+1$ a "jump" is to be made in executing instruction 12, 13, or 14 of Fig. 2, control wire $\underline{C}_r = 1$ and $\underline{C}_t = 0$, and the address $\underline{X}$ of bin $\underline{X}$ referred to in the "jump" instruction OBEY X (or OBEY X IF MINUS or OBEY X IF $\underline{a}_{15}$ IS 1) occupies wires $\underline{s}_4, \underline{s}_5, \ldots, \underline{s}_{15}$ (as will be explained in the following section). Since $\underline{C}_t = 0$, no signal can pass from $\underline{a}'_i$ onto

$\underline{s}_i$ through the cell's rightmost conjunction element; and since $\underline{C}_r = 1$, no signal can pass from $\underline{a}_i'$ to $\underline{r}_i'$ through the conjunction element to the left of $\underline{a}_i'$. But the lower left-hand conjunction element permits any bit of information on $\underline{s}_i$ to pass onto $\underline{r}_i'$ at time $\tau = 2\underline{k}+1$ and to occupy $\underline{a}_i'$ at time $\tau = 2\underline{k}+2$. Hence the address $\underline{X}$ of bin $\underline{X}$ referred to in the "jump" instruction OBEY X will be contained in the ADDRESS COUNTER at time $\tau = 2\underline{k}+2$. At that even numbered moment both $\underline{S}_t$ and $\underline{C}_t$ are automatically activated, which makes bin $\underline{X}$ of the parallel storage transmit its contents onto the trunk at time $\tau = 2\underline{k}+2$, and causes the ADDRESS DECODER to contain, at the following moment $\tau = 2\underline{k}+3$, the address of bin $\underline{X}+1$.

6.2. CONTROL CLOCK.

We wish to be able to start the machine at any time, and we want control wires $\underline{S}_t$ and $\underline{C}_t$ to be automatically activated at every even numbered moment while the machine is solving a problem. Our device for accomplishing this function is the CONTROL CLOCK, which occupies the lower right-hand part of Fig. 9. The behavior of the right-hand delay circuit, which may be described as a "blocking oscillator circuit", is described by the equation $\underline{g}(\tau) \equiv (\tau \equiv 0 \bmod 2)$. Thus $\underline{g}$ is activated at $\tau = 0, 2, 4, \ldots, 2\underline{k}, \ldots$ independently of anything that may happen elsewhere in the machine.

The left-hand delay circuit operates somewhat differently. If the wire $\underline{f}$ is inactive, then activating the START KEY at time τ will activate $\underline{h}$. At time $\tau+1$ the signal will emerge from the delay element above, and if $\underline{f}$ is still inactive, the signal will pass onto $\underline{h}$ again and up again to the delay element. Thus, so long as $\underline{f} = 0$, activating the START KEY at time τ will activate $\underline{h}$ at times $\tau, \tau+1, \tau+2, \ldots$.

The two delay circuits work together to produce the following result. So long as $\underline{f} = 0$, if the START KEY is activated at either time $\tau = 2\underline{k}-1$ or $2\underline{k}$, $\underline{h}$ will be activated at times $2\underline{k}$, $2\underline{k}+1$, $2\underline{k}+2, \ldots$. The other input wire $\underline{g}$ is activated by the right-hand delay circuit at times $2\underline{k}$, $2\underline{k}+2$, $2\underline{k}+4, \ldots$. Hence the output wire controlling $\underline{C}_t$ and $\underline{S}_t$ is activated at times $2\underline{k}$, $2\underline{k}+2$, $2\underline{k}+4$, and so on. Once started, the computer's activity is cyclic, with control wires $\underline{S}_t$ and $\underline{C}_t$ activated at every even part of the cycle, starting at the moment the START KEY is activated if that is done at an even numbered moment, or at the following moment if the START KEY was activated at an odd numbered moment.

We wish also to be able to stop the machine, both manually and by instruction 1 of Fig. 2. To stop the machine we must activate $\underline{f}$, which will prevent any signal from the upper delay element passing onto $\underline{h}$, thus clearing the left-hand delay circuit. As Fig. 9 shows, $\underline{f}$ can be activated either by activating the STOP KEY or by activating $\underline{d}_3$ but not $\underline{d}_0$, $\underline{d}_1$, or $\underline{d}_2$. Hence the operand 0001 signals the machine to stop. This circuit is part of the OPERAND DECODER, which is discussed in the following subsection.

6.3. OPERAND DECODER.

We have shown in Sections 4 and 5 how the STORAGE and the ARITHMETIC UNIT function when their various control wires are activated, and we have listed in Fig. 2 the various different instruction words that activate different sets of control wires. It is the operand or first four binary digits of an instruction word which specifies which control wires are to be activated for the instruction to be executed. The OPERAND DECODER which occupies the left-hand part of Fig. 9 is a switching mechanism which activates the proper set of control wires when it receives the four digits of an operand.

The four wires $\underline{d}_0$, $\underline{d}_1$, $\underline{d}_2$, $\underline{d}_3$ are connected to various threshold elements in the way indicated. That the desired functional connections are realized by the OPERAND DECODER is easily verified. For example, the control wire $\underline{A}_r$, which must be activated for the execution of instruction 4, 5, 6, 7, 8, or 9 of Fig. 2, is activated by any of their operands, as described by the following equation:

$$\underline{A}_r \equiv (\underline{\bar{d}}_0\underline{d}_1\underline{\bar{d}}_2\underline{\bar{d}}_3 \vee \underline{\bar{d}}_0\underline{d}_1\underline{\bar{d}}_2\underline{d}_3 \vee \underline{\bar{d}}_0\underline{d}_1\underline{d}_2\underline{\bar{d}}_3 \vee \underline{\bar{d}}_0\underline{d}_1\underline{d}_2\underline{d}_3 \vee \underline{d}_0\underline{\bar{d}}_1\underline{\bar{d}}_2\underline{\bar{d}}_3 \vee \underline{d}_0\underline{\bar{d}}_1\underline{\bar{d}}_2\underline{d}_3).$$

That equation can be simplified to

$$\underline{A}_r \equiv (\underline{\bar{d}}_0\underline{d}_1 \vee \underline{d}_0\underline{\bar{d}}_1\underline{\bar{d}}_2),$$

which is obviously realized by the address decoder as diagrammed in Fig. 9. The functional connections between operands and various sets of control wires could be specified in many different ways: the present arrangement was selected to permit simplifications of the kind indicated.

6.4. Operation of the CONTROL.

Four parts of our computer are represented by blocks in Fig. 1, and each has been explained in detail. We have now to explain the rest of the CONTROL, which consists of sixteen conjunction elements and sixteen delay elements, together with the wires connecting them to the other parts of the computer. Each component $\underline{t}_i$ of the trunk is connected to an input wire of one of these conjunction elements, whose other input wire is connected to control wire $\underline{C}_t$. The output wires $\underline{t}_i''$ of those conjunction elements lead to separate delay elements, whose output wires lead to $\underline{d}_0$, $\underline{d}_1$, $\underline{d}_2$, $\underline{d}_3$ and $\underline{s}_4$, $\underline{s}_5$, ..., $\underline{s}_{15}$.

The functioning of these parts can best be explained by showing how they operate when the machine is executing an instruction under the direction of the CONTROL. When the machine is engaged in solving a problem, at any even numbered moment $\tau = 2\underline{k}$ control wires $\underline{S}_t$ and $\underline{C}_t$ are automatically activated, making the ADDRESS COUNTER send the address of some storage bin down wires $\underline{s}_4$, $\underline{s}_5$, ..., $\underline{s}_{15}$ to the STORAGE. The STORAGE is thereby caused to transmit the instruction word from that bin onto the trunk. The sixteen digits of that instruction word pass through the sixteen conjunction elements above (since $\underline{C}_t(2\underline{k}) = 1$) to occupy the sequence of wires $\underline{t}''$. At the following moment $\tau = 2\underline{k}+1$ (an odd cycle) the sixteen bits of the instruction word emerge from the sixteen delay elements. The first four bits (its operand) pass along $\underline{d}_0$, $\underline{d}_1$, $\underline{d}_2$, $\underline{d}_3$ to the OPERAND DECODER, and the last twelve bits (its address) pass along $\underline{s}_4$, $\underline{s}_5$, ..., $\underline{s}_{15}$ either up to the ADDRESS COUNTER, if its operand causes the OPERAND DECODER to activate control wire $\underline{C}_r$, or down to the STORAGE if its operand causes the OPERAND DECODER to stimulate either control wire $\underline{S}_t$, $\underline{S}_{rt}$, or $\underline{S}_{rp}$.

Thus we see that at every even cycle the computer brings a new instruction word from STORAGE onto its trunk, and at each following odd cycle the computer executes the instruction enjoined by that instruction word. We will now illustrate this process.

Consider the actual sequence of occurrences when the computer begins a routine whose first two instructions TRANSFER 101 and OBEY 6 IF $\underline{a}_{15}$ IS 1 are in bins 0 and 1, respectively.[8] Now if the ADDRESS COUNTER contains all zeros, and we activate the START KEY at either time $\mathcal{T} = 2\underline{k}-1$ or $\mathcal{T} = 2\underline{k}$, the machine will begin its run at $\mathcal{T} = 2\underline{k}$, when $\underline{S}_t = \underline{C}_t = 1$, and the ADDRESS COUNTER transmits 00...00, the address of bin 0, along wires $\underline{s}_4$, $\underline{s}_5$, ..., $\underline{s}_{15}$, to the STORAGE. Bin 0 transmits its contents 0110000001100101 onto the trunk, up through the conjunction elements onto $\underline{t}''_1$. Then at $\mathcal{T} = 2\underline{k}+1$ those bits emerge from the delay elements, the operand 0110 going along $\underline{d}_0$, $\underline{d}_1$, $\underline{d}_2$, $\underline{d}_3$ to cause the OPERAND DECODER to activate control wires $\underline{S}_t$ and $\underline{A}_r$. At the same time the address 000001100101 goes down $\underline{s}_4$, $\underline{s}_5$, ..., $\underline{s}_{15}$ to the STORAGE, which transmits the number $\underline{x}$ from bin 101 onto the trunk, and then into the cells of the ARITHMETIC UNIT, since $\underline{A}_r$ is activated. At $\mathcal{T} = 2\underline{k}+2$ we have $\underline{S}_t = \underline{C}_t = 1$ again, and the ADDRESS COUNTER transmits 00...01, the address of bin 1, along $\underline{s}_4$, $\underline{s}_5$, ..., $\underline{s}_{15}$ to the STORAGE, which transmits its contents 1110000000000110 onto the trunk, up through the conjunction elements onto $\underline{t}''$. Then at $\mathcal{T} = 2\underline{k}+3$ those bits emerge from the delay elements, 1110 along $\underline{d}_0$, $\underline{d}_1$, $\underline{d}_2$, $\underline{d}_3$ to the OPERAND DECODER, and 00...0110 onto $\underline{s}_4$, $\underline{s}_5$, ..., $\underline{s}_{15}$. Now what control wires the OPERAND DECODER activates will depend upon the present contents of the ARITHMETIC UNIT, which contains the number $\underline{x}_0\underline{x}_1\ldots\underline{x}_{15}$. If the rightmost digit $\underline{x}_{15}$ of that number is 1, $\underline{a}_{15} = 1$, and control wire $\underline{C}_r$ is stimulated by the OPERAND DECODER. In this case the address 00...0110 now occupying wires $\underline{s}_4$, $\underline{s}_5$, ..., $\underline{s}_{15}$ enters the ADDRESS COUNTER, which at the following time $\mathcal{T} = 2\underline{k}+4$ will transmit it down to the STORAGE to cause the instruction word in bin 6 to pass onto the trunk. But if the rightmost digit $\underline{x}_{15}$ is 0, $\underline{a}_{15} = 0$, and the OPERAND DECODER activates no control wires. In this case the address in the ADDRESS COUNTER remains 00...010, and at the following time $\mathcal{T} = 2\underline{k}+4$ it is transmitted down to the STORAGE to cause the instruction word in bin 2 to pass onto the trunk.

Footnotes

1. The writing of this paper was done under the sponsorship of the Burroughs Corporation, Research Center, Paoli, Pennsylvania.

   The logical design and analysis is primarily the work of the first author (A.W.B.) and the exposition is largely the work of the second author (I.M.C.). Thanks are due to Dr. R. L. Cartwright for helpful criticisms and suggestions.

2. Some logical nets involving several primitive elements may be physically realized by devices no more complicated than those required for the physical realization of a single primitive element. For example, an electronic circuit using a single multigrid tube can realize a conjunction element.

3. The threshold elements here defined are different from those discussed in either [5] or [6] .

4. Forty bits is a word length more typical of actual computers.

5. The complexity of the Address Decoder could be reduced by using a less straightforward design, as explained in [3] .

6. An actually infinite serial storage is not a net in the sense of [2] . But the serial storage is so connected to the machine that the whole net has the property

that at any arbitrary time only a finite number of its wires have changed from their initial states. Hence at no time is the net presumed to contain an infinite amount of information, and it can therefore be regarded as a finite but indefinitely expansible net.

7. The serial storage may be realized by a magnetic tape (or set of tapes) with sixteen parallel channels of information. Here a single cell corresponds to a region of the tape which is or is not magnetized, and the operation of shifting the contents of the bins forward or backward is realized by moving the tape mechanically. On this interpretation the serial storage represents a relatively rapid way in which information can pass between the computing machine and its operator.

8. See Appendix C for a complete routine of which this is a segment.

Bibliography

[1] Burks, Arthur W., Herman H. Goldstine, and John von Neumann, Preliminary Discussion of the Logical Design of an Electronic Computing Instrument, Part I, Vol. I, The Institute for Advanced Study, 1946.

[2] Burks, Arthur W., and Jesse B. Wright, "Theory of Logical Nets," Proceedings of the I.R.E., Vol. 41, 1953, pp. 1357-1365.

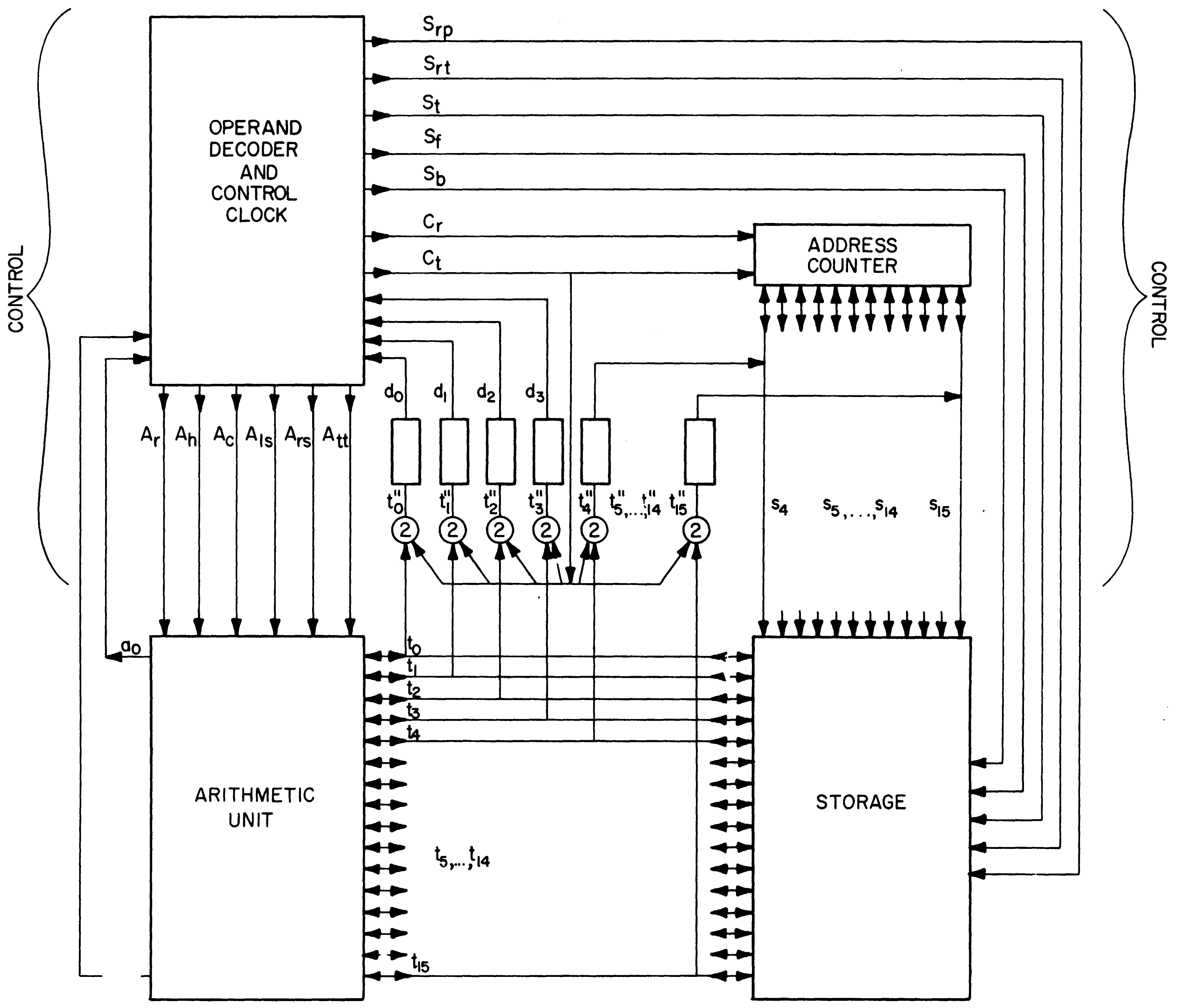
[3] Burks, Arthur W., Robert McNaughton, Carl H. Pollmar, Don W. Warren, and Jesse B. Wright, Complete Decoding Nets: General Theory and Minimality, ERI Report 1828-1-T (Burroughs Corporation Research Center, Paoli, Pennsylvania), Ann Arbor, 15 July 1954. To be published in the journal of the Society for Industrial and Applied Math.

[4] Copi, Irving M., Symbolic Logic, New York, 1954.

[5] Kleene, S. C., "Representation of events in nerve nets and finite automata," RM-704, RAND Corporation, 1951.

[6] McCulloch, Warren L., and Walter Pitts, "A logical calculus of the ideas immanent in neuron activity," Bull. of Math. Biophysics, Vol. 5, 1943, pp. 115-133.

[7] Patterson, George W., "Logical Syntax and Transformation Rules," Proceedings of a Second Symposium on Large-Scale Calculating Machinery (Cambridge, Mass., September 13-16, 1949), Cambridge, 1951, pp. 125-133.


CONTROL
OPERAND DECODER AND CONTROL CLOCK
S_{rp}
S_{rt}
S_t
S_f
S_b
C_r
C_t
ADDRESS COUNTER
CONTROL
A_r A_h A_c A_{ls} A_{rs} A_{tt}
d_0 d_1 d_2 d_3
t''_0 t''_1 t''_2 t''_3 t''_4 $t''_5,\ldots,t''_{14}$ t''_{15}
2
s_4 $s_5,\ldots,s_{14}$ s_{15}
a_0
t_0 t_1 t_2 t_3 t_4
ARITHMETIC UNIT
$t_5,\ldots,t_{14}$
t_{15}
STORAGE

BLOCK DIAGRAM OF COMPUTER

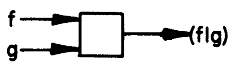
Fig. 1

	INSTRUCTION	OPERAND $d_0d_1d_2d_3$	OPERAND DECODER OUTPUT WIRES ACTIVATED AT $\tau = 2k + 1$
1.	STOP	0001	
2.	SERIAL FORWARD (Move contents of serial storage one bin forward)	0010	S_f
3.	SERIAL BACKWARD (Move contents of serial storage one bin backward)	0011	S_b
4.	ADD X (Add the number in storage bin X to the number in the arithmetic unit)	0100	$S_tA_rA_h$
5.	SUBTRACT X (Subtract the number in storage bin X from the number in the arithmetic unit)	0101	$S_tA_rA_hA_c$
6.	TRANSFER X (Transfer the number in storage bin X to the arithmetic unit)	0110	S_tA_r
7.	TRANSFER COMPLEMENT X (Transfer the negative of the number in storage bin X to the arithmetic unit)	0111	S_tA_r A_c
8.	DOUBLE (Double the number in the arithmetic unit)	1000	A_r A_{ls}
9.	HALVE (Halve the number in the arithmetic unit)	1001	A_r A_{rs}
10.	STORE IN X (Replace the number in storage bin X by the number in the arithmetic unit)	1010	$S_{rt}A_{tt}$
11.	SUBSTITUTE IN X (Replace the rightmost 12 digits of the number in storage bin X by the corresponding digits of the number in the arithmetic unit)	1011	$S_{rp}A_{tt}$
12.	OBEY X (Execute next the instruction in storage bin X)	1100	C_r
13.	OBEY X IF MINUS (If $a_0 = 1$, then execute next the instruction in storage bin X)	1101	$a_0 \supset C_r$
14.	OBEY X IF a_{15} IS 1 (If $a_{15} = 1$, then execute next the instruction in storage bin X)	1110	$a_{15} \supset C_r$

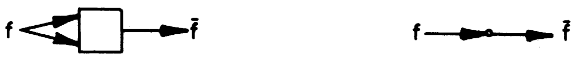
(At every even cycle $\tau = 2k$ when the machine is operating, $d_0 = d_1 = d_2 = d_3 = 0$, and wires S_t and C_t are activated.)

TABLE OF INSTRUCTIONS

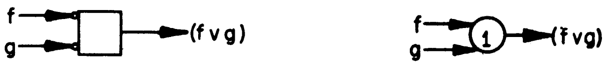
Fig. 2



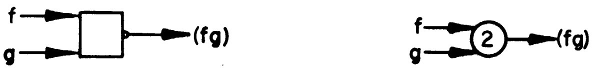
a. **STROKE ELEMENT:** $f|g$



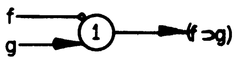
b. **NEGATION ELEMENT:** $\bar{f} =_{df} (f|f)$



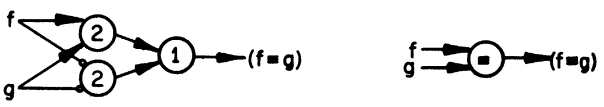
c. **DISJUNCTION ELEMENT:** $(f \vee g) =_{df} (\bar{f}|\bar{g})$



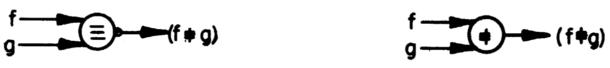
d. **CONJUNCTION ELEMENT:** $(fg) =_{df} (\overline{f|g})$



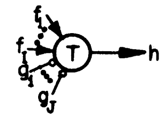
e. **MATERIAL IMPLICATION ELEMENT:** $(f \supset g) =_{df} (\bar{f} \vee g)$



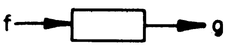
f. **EQUIVALENCE ELEMENT:** $(f \equiv g) =_{df} [(fg) \vee (\bar{f}\bar{g})]$



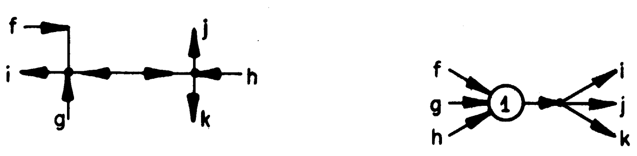
g. **INEQUIVALENCE ELEMENT:** $(f \not\equiv g) =_{df} [(f\bar{g}) \vee (\bar{f}g)]$



h. **GENERALIZED THRESHOLD ELEMENT**

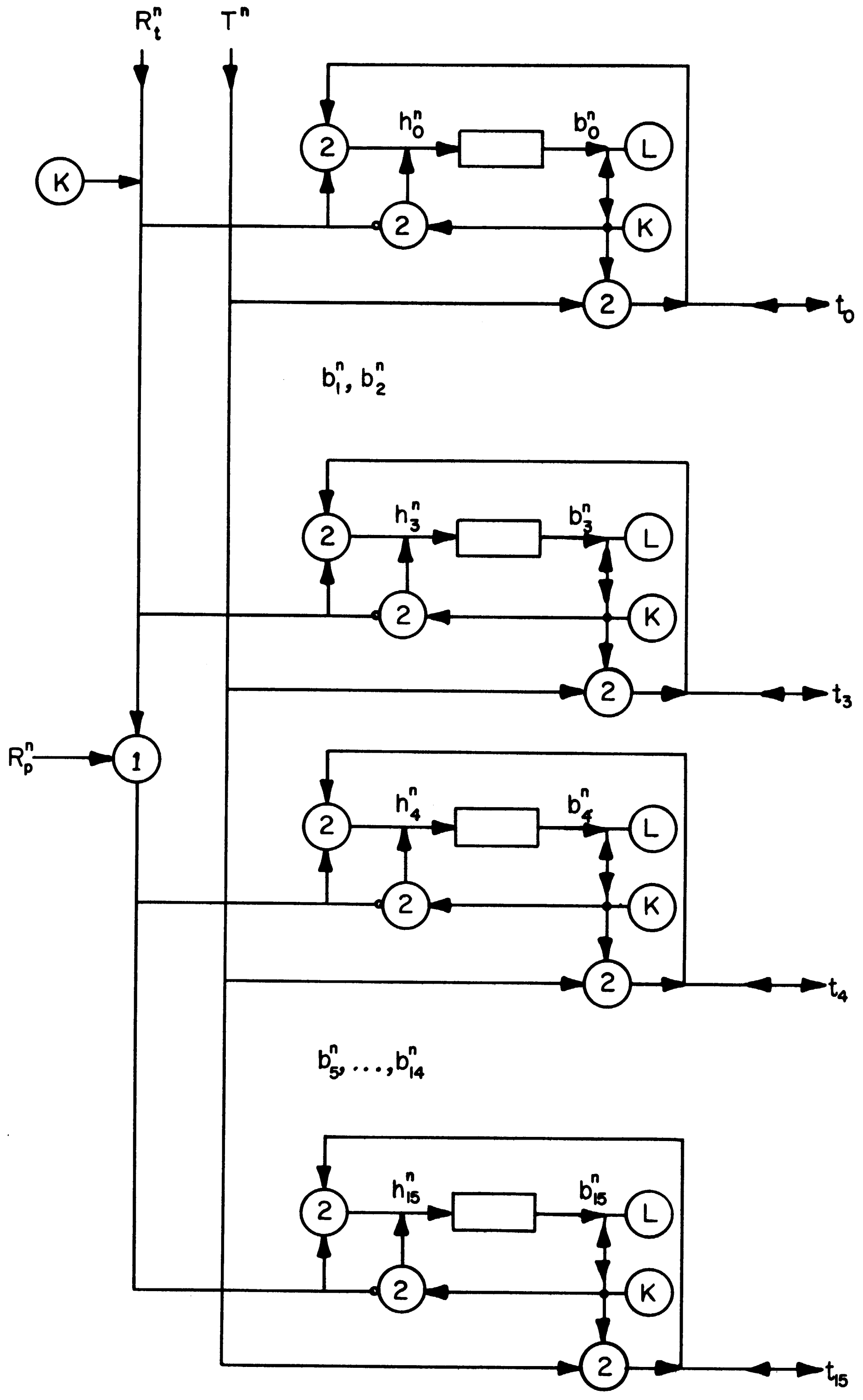


i. **DELAY ELEMENT:** $g(0) = 0,\ g(\tau+1) = f(\tau)$ FOR $\tau = 0,1,2,\ldots$



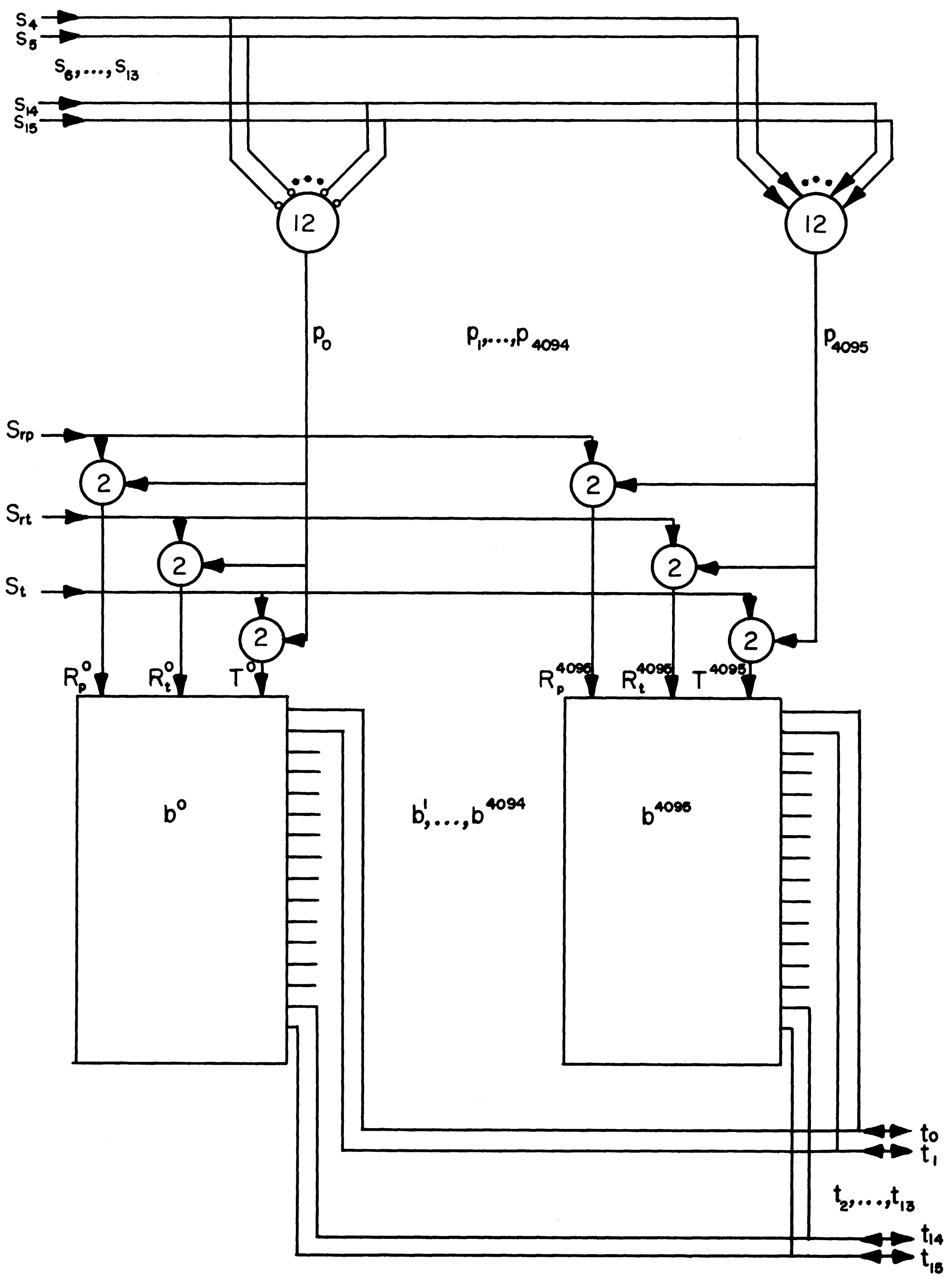
j. **MULTIPLE JUNCTION ABBREVIATING A MULTIPLE INPUT DISJUNCTION ELEMENT:** $(f \vee g \vee h) = i = j = k$.

ELEMENTS
Fig. 3



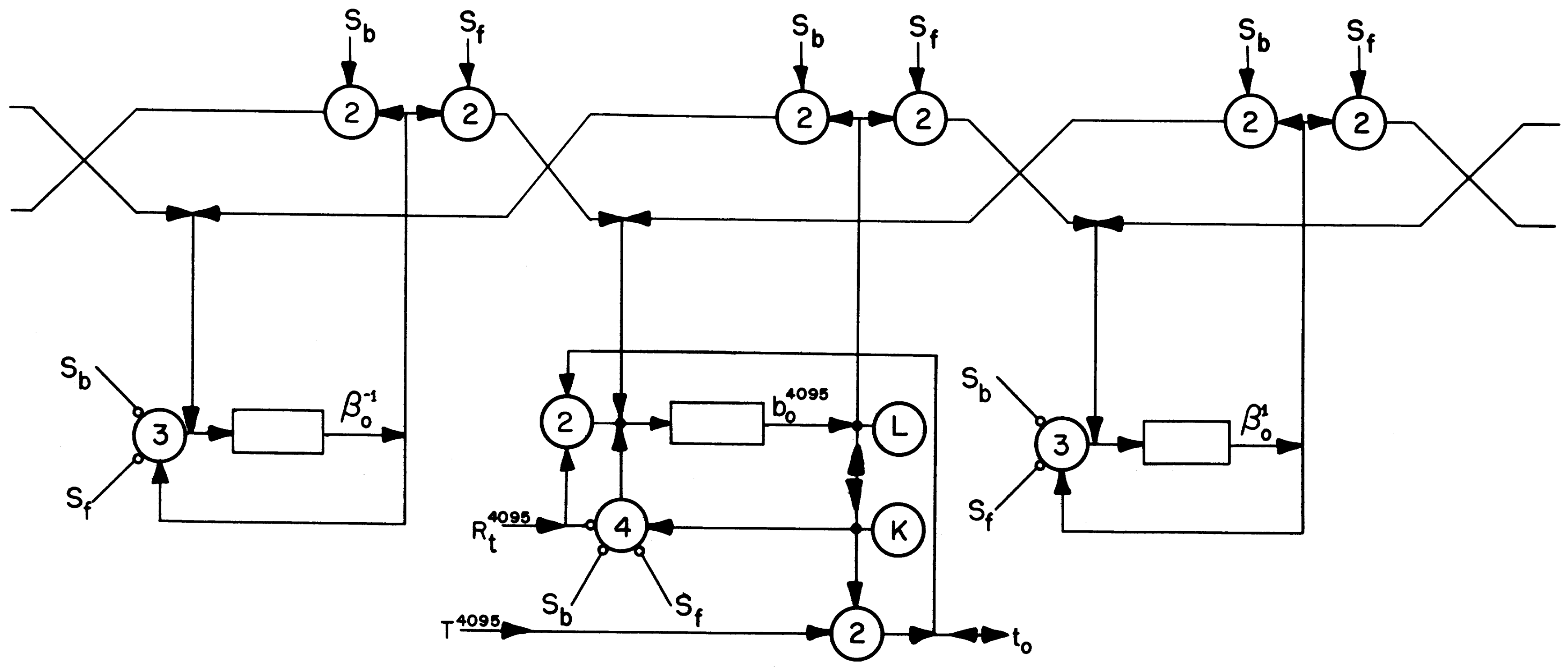
BIN OF PARALLEL STORAGE

Fig. 4



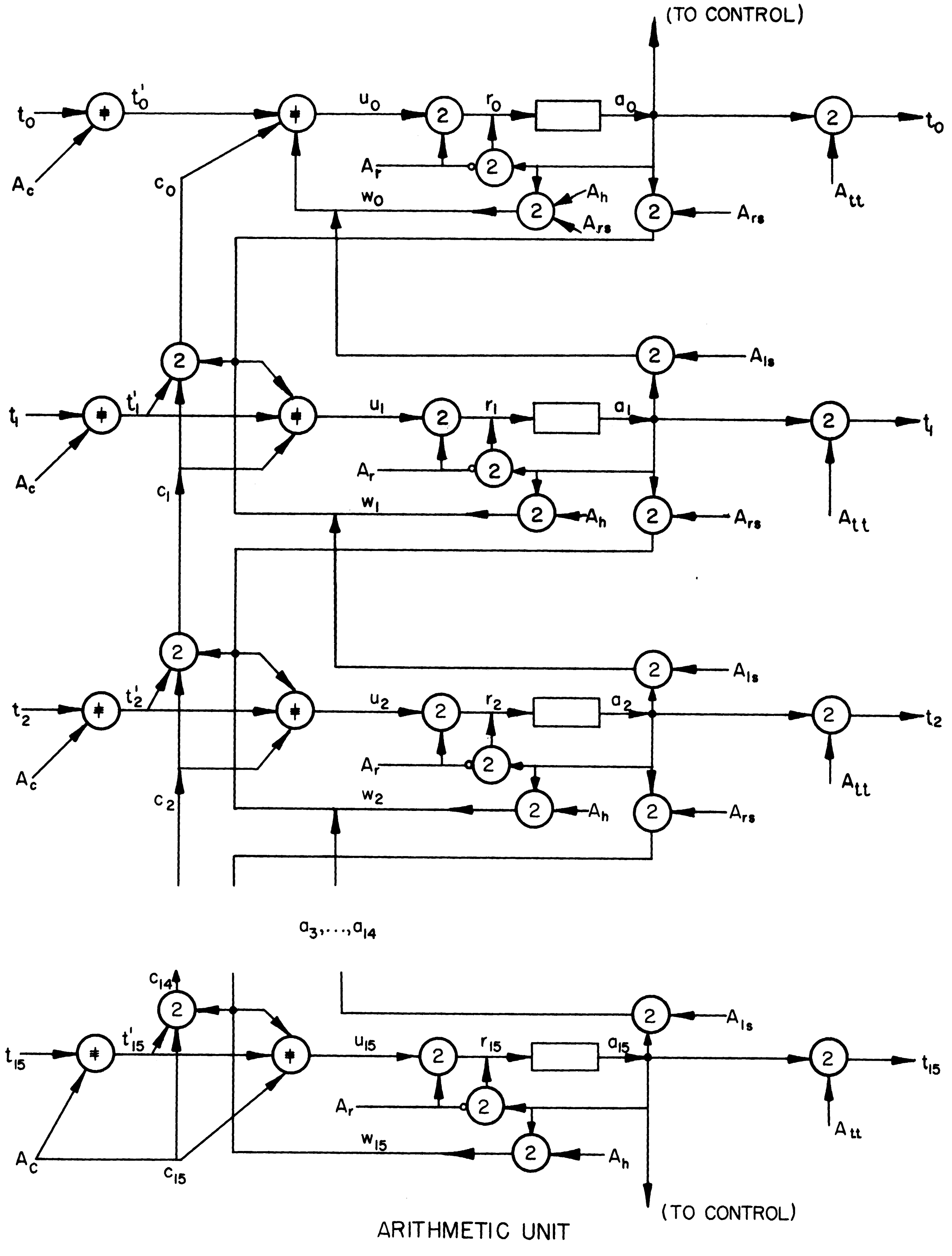
PARALLEL STORAGE

Fig. 5



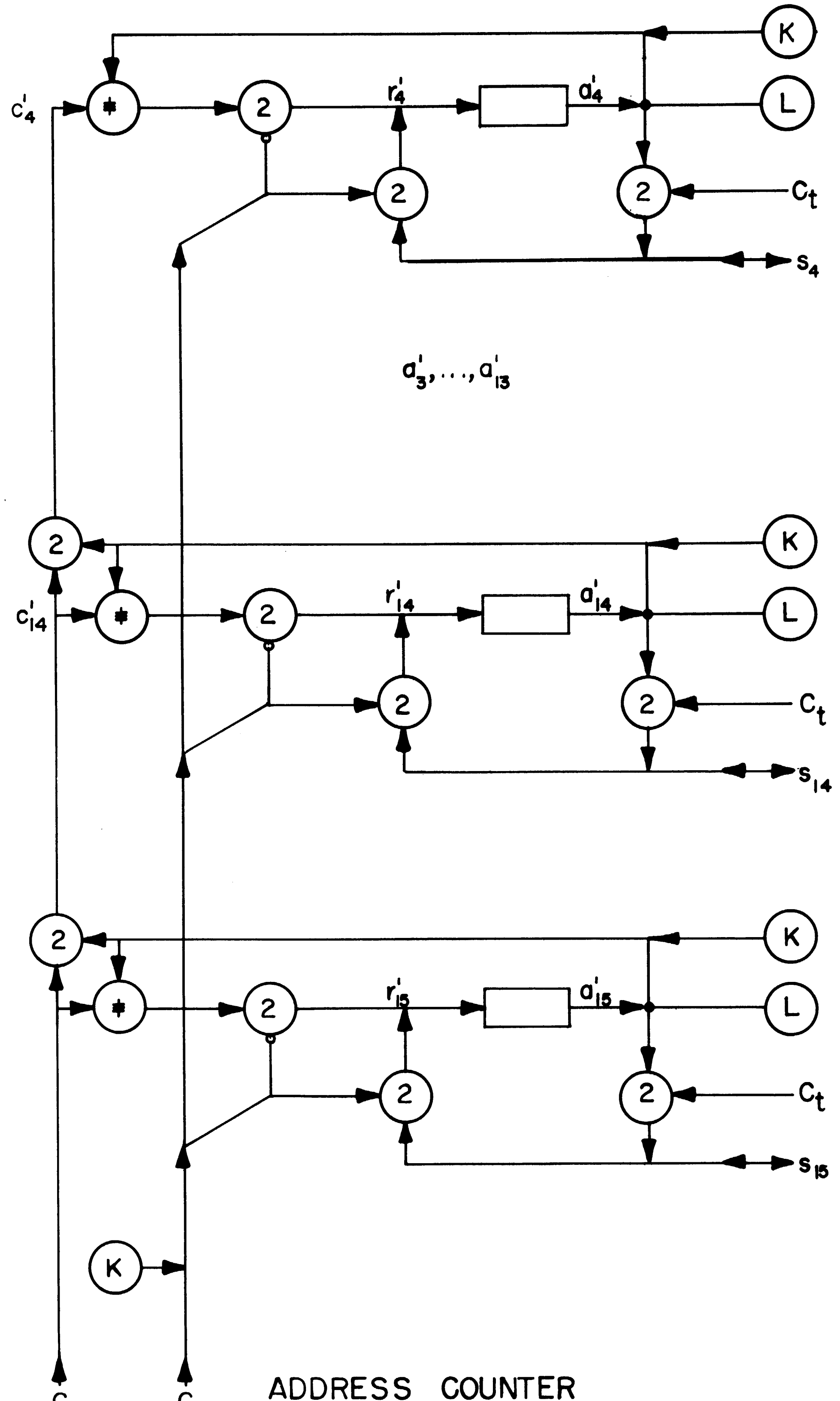
SERIAL STORAGE

Fig. 6



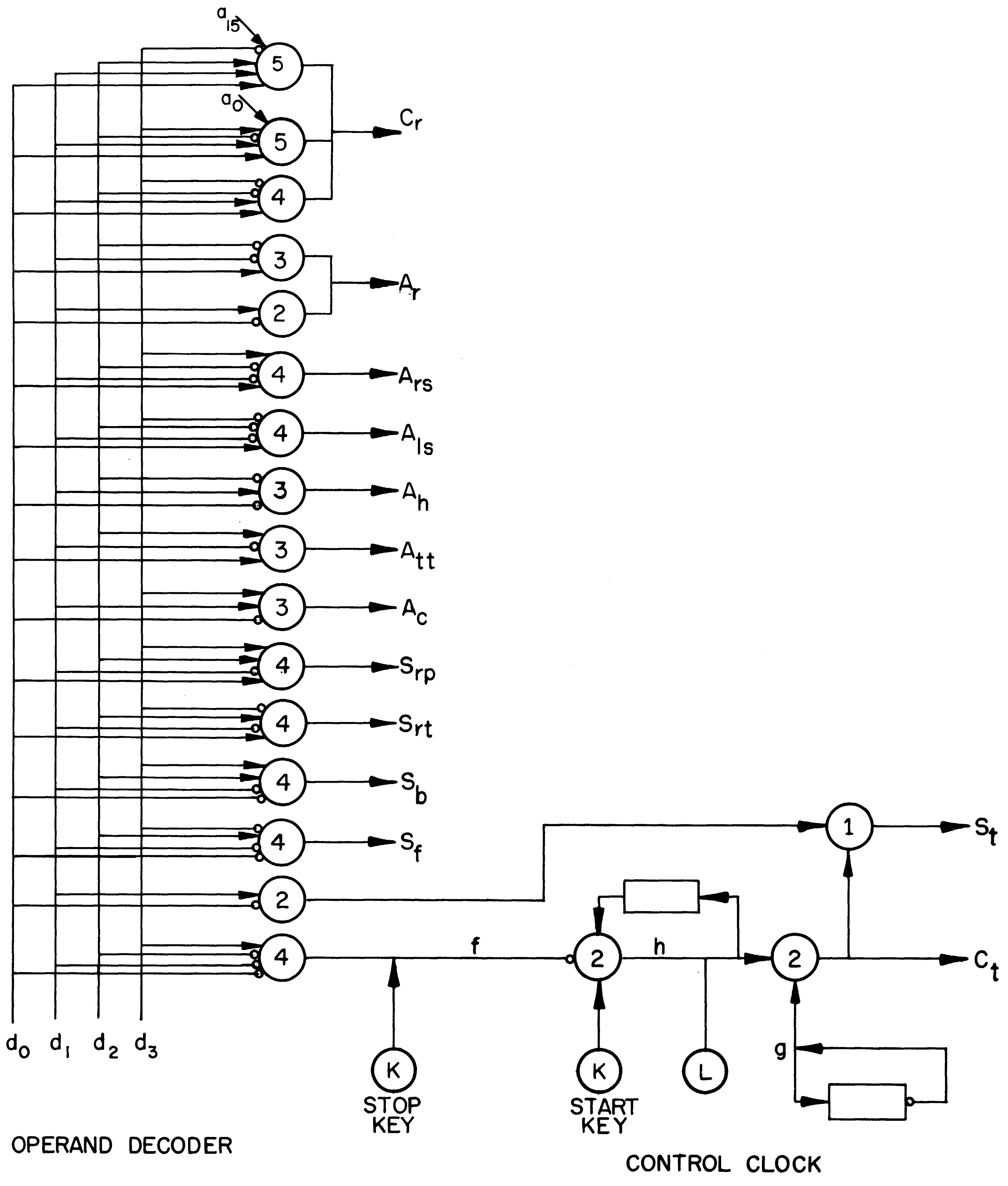
ARITHMETIC UNIT

Fig. 7



ADDRESS COUNTER

Fig. 8



OPERAND DECODER AND CONTROL CLOCK

Fig. 9

Appendix A. Some Operations of the ARITHMETIC UNIT

In this appendix we shall illustrate the actual machine procedure in executing arithmetic instructions.

1. Subtraction.

If the instruction SUBTRACT X is executed, $\underline{S}_t = \underline{A}_r = \underline{A}_h = \underline{A}_c = 1$ and all other control wires are in state 0. Let us suppose that the number $\underline{x}_0\underline{x}_1\ldots\underline{x}_{15}$ is in storage bin $\underline{X}$ and the number $\underline{y}_0\underline{y}_1\ldots\underline{y}_{15}$ is in the cells of the ARITHMETIC UNIT.

Since $\underline{S}_t = 1$, the number in bin $\underline{X}$ will pass onto the trunk, whence $\underline{t}_i \equiv \underline{x}_i$. Each trunk component $\underline{t}_i$ is an input wire to an inequivalence element (at the left of Fig. 7) whose other input wire is $\underline{A}_c$, and whose output wire is $\underline{t}'_i$, whence $\underline{t}'_i \equiv (\underline{t}_i \not\equiv \underline{A}_c)$. Now $\underline{A}_c = 1$, so $\underline{t}'_i \equiv \overline{\underline{t}}_i$ or $\underline{t}'_i \equiv \overline{\underline{x}}_i$.

By assumption, $\underline{a}_i \equiv \underline{y}_i$, and since $\underline{A}_h = 1$, each bit $\underline{y}_i$ passes through the lower left-hand conjunction element of its cell onto wire $\underline{w}_i$, whence $\underline{w}_i \equiv \underline{y}_i$.

We now observe that $\underline{t}'_i$, $\underline{w}_i$, and $\underline{c}_i$ are the three input wires of an inequivalence element whose output wire is $\underline{u}_i$, whence $\underline{u}_i \equiv (\underline{t}'_i \not\equiv \underline{w}_i \not\equiv \underline{c}_i)$ or $\underline{u}_i \equiv (\overline{\underline{x}}_i \not\equiv \underline{y}_i \not\equiv \underline{c}_i)$. We also observe that where $\underline{i} > 0$, $\underline{t}'_i$, $\underline{w}_i$, and $\underline{c}_i$ are the three input wires of a threshold-2 element whose output wire is $\underline{c}_{i-1}$, whence $\underline{c}_{i-1} \equiv (\underline{t}'_i\underline{w}_i \vee \underline{t}'_i\underline{c}_i \vee \underline{w}_i\underline{c}_i)$ or $\underline{c}_{i-1} \equiv$ $(\underline{\bar{x}}_i\underline{y}_i \vee \underline{\bar{x}}_i\underline{c}_i \vee \underline{y}_i\underline{c}_i)$, for $\underline{i} > 0$. Now $\underline{c}_{15} \equiv \underline{A}_c$ and $\underline{A}_c = 1$, whence $\underline{c}_{15} = 1$, and with this condition added, the two preceding equations for $\underline{u}_i$ and $\underline{c}_{i-1}$ conform to the general recursive definition of subtraction, whence the number whose bits occupy wires $\underline{u}_i$, say $\underline{z}_0\underline{z}_1\cdots\underline{z}_{15}$, is congruent modulo 2 to the difference $\underline{y}_0\underline{y}_1\cdots\underline{y}_{15} - \underline{x}_0\underline{x}_1\cdots\underline{x}_{15}$.

Since $\underline{A}_r = 1$, the bit occupying $\underline{a}_i$ cannot pass through the conjunction 2 element below $\underline{r}_i$, but the bit $(\underline{z}_i)$ occupying wire $\underline{u}_i$ can pass through the conjunction element to the left of $\underline{r}_i$ and thus pass onto $\underline{r}_i$. Hence, if SUBTRACT X is executed at time τ, a number congruent modulo 2 to the result of subtracting the number in bin $\underline{X}$ from the number in the ARITHMETIC UNIT will occupy wires $\underline{r}_i$ at time τ and will occupy the cells $\underline{a}_i$ at time $\tau + 1$.

2. Doubling.

A number in binary notation is doubled by moving its binary point one position to the right, which amounts to shifting each of its digits one position to the left. If the instruction DOUBLE is executed, $\underline{A}_r = \underline{A}_{ls} = 1$ and all other control wires are 0. Suppose the number $\underline{y}_0.\underline{y}_1\cdots\underline{y}_{15}$ occupies the cells $\underline{a}_i$ of the ARITHMETIC UNIT.

Since $\underline{S}_t = \underline{A}_{tt} = 0$, every component of the trunk $\underline{t}_i = 0$; and since $\underline{A}_c = 0$, each $\underline{t}'_i = 0$ because it is the output wire of an inequivalence element whose input wires are equivalent (both 0). Since $A_c = 0$, $c_{15} = 0$, and every $c_i = 0$, since each c_i for $i < 15$ is the output wire of a three-input threshold-2 element, two of whose inputs are 0.

Since $A_h = 0$ (and $A_{rs} = 0$, which is relevant in cell a_0) no signal from a_i can pass through the lower left-hand conjunction element onto wire w_i. Instead $w_{15} = 0$ and each wire w_i for $i < 15$ is occupied by the bit y_{i+1} from a_{i+1}, which can pass up to w_i through the conjunction element whose input wires are a_{i+1} and A_{ls}, since $A_{ls} = 1$.

Now each u_i contains the same bit as w_i, for u_i is the output wire of an inequivalence element whose three input wires are t_i', c_i, and w_i, the first two of which are in state 0. Since $A_r = 1$, $r_i \equiv u_i$, whence the number occupying wires r_i is $y_1.y_2...y_{15}0$. Hence if DOUBLE is executed at time τ, a number congruent modulo 2 to twice the number in the ARITHMETIC UNIT will occupy wires r_i at time τ and will occupy the cells a_i at time $\tau+1$.

Appendix B. Machine Solution of a Simple Problem.

To program the computer for evaluation of the polynomial $2x+y-z$ for a particular set of values of its variables, say 1/4, 3/8, and 5/16, we proceed as follows. First, we store the indicated values of x, y, and z in bins 11, 12, and 13. Then we decide that a straightforward way for the machine to solve the problem is to have it transfer x to the ARITHMETIC UNIT, double it, add to that result the number y, and then subtract from that result the number z. At that time the desired solution will be in the ARITHMETIC UNIT. Standard procedure is to have the machine store its answer in a bin of the parallel storage before it stops; we select bin 14 for this purpose.

Now the routine is inserted into the computer: the first instruction word in bin 0, the second in bin 1, and so on. The routine for this problem is written below, on the left informally, on the right coded for insertion into the machine.

```
bin 0:  TRANSFER 11          0110000000001011
bin 1:  DOUBLE               1000000000000000
bin 2:  ADD 12               0100000000001100
bin 3:  SUBTRACT 13          0101000000001101
bin 4:  STORE IN 14          1010000000001110
bin 5:  STOP                 0001000000000000
  .
  .
  .
```

bin 11:	$\underline{x}$	0010000000000000
bin 12:	$\underline{y}$	0011000000000000
bin 13:	$\underline{z}$	0010100000000000
bin 14:	(to contain solution:	0100100000000000)

Having completed preparing the machine by clearing its ADDRESS COUNTER to 00...0, we activate the START KEY at time $\mathcal{T} = 2\underline{k}-1$ or $\mathcal{T} = 2\underline{k}$, and the computation begins at time $\mathcal{T} = 2\underline{k}$. At $\mathcal{T} = 2\underline{k}$ the ADDRESS COUNTER transmits its contents along $\underline{s}_4$, $\underline{s}_5$, ..., $\underline{s}_{15}$ to the STORAGE, which sends the instruction word from bin 0 onto the trunk and up through the sixteen conjunction elements to $\underline{t}''$. At $\mathcal{T} = 2\underline{k}+1$ the instruction word initially in bin 0 emerges from the delay elements above $\underline{t}''$, its operand 0110 going via $\underline{d}_0$, ..., $\underline{d}_3$ to the OPERAND DECODER, and its address 0...01011 going via $\underline{s}_4$, $\underline{s}_5$, ..., $\underline{s}_{15}$ to the STORAGE. These cause bin 11 to transmit its contents 0010...0 onto the trunk and cause the ARITHMETIC UNIT to receive that number from the trunk into its cells $\underline{a}_i$. At $\mathcal{T} = 2\underline{k}+2$ the ADDRESS COUNTER transmits its contents (now 0...01) along $\underline{s}_4$, $\underline{s}_5$, ..., $\underline{s}_{15}$ to the STORAGE, which sends the instruction word from bin 1 onto the trunk and up through the conjunction elements to $\underline{t}''$. At every even cycle a new instruction word goes through the trunk up to the delay elements, and at the following odd cycle that instruction is executed. Finally, at $\mathcal{T} = 2\underline{k}+11$ the machine stops.

The machine's activity during this run can be shown in tabular form. All storage bins retain their initial contents throughout the run except bin 14 which receives the solution at time $\tau = 2\underline{k}+9$. Significant changes occur in the ADDRESS COUNTER, in the ARITHMETIC UNIT, and in wire sets $\underline{t}''$; $\underline{d}_0, \ldots, \underline{d}_3$; and $\underline{s}_4, \underline{s}_5, \ldots, \underline{s}_{15}$. These are all shown in the following table.

Time τ	Address Counter	$\underline{t}''_0, \underline{t}''_1, \ldots, \underline{t}''_{15}$	$\underline{d}_0, \ldots, \underline{d}_3$	$\underline{s}_4, \underline{s}_5, \ldots, \underline{s}_{15}$	Arithmetic Unit
$2\underline{k}$	0	01100...01011	0000	0...00000	0
$2\underline{k}+1$	1	00000...00000	0110	0...01011	$\underline{x}$
$2\underline{k}+2$	1	10000...00000	0000	0...00001	$\underline{x}$
$2\underline{k}+3$	2	00000...00000	1000	0...00000	$2\underline{x}$
$2\underline{k}+4$	2	01000...01100	0000	0...00010	$2\underline{x}$
$2\underline{k}+5$	3	00000...00000	0100	0...01100	$2\underline{x}+\underline{y}$
$2\underline{k}+6$	3	01010...01101	0000	0...00011	$2\underline{x}+\underline{y}$
$2\underline{k}+7$	4	00000...00000	0101	0...01101	$2\underline{x}+\underline{y}-\underline{z}$
$2\underline{k}+8$	4	10100...01110	0000	0...00100	$2\underline{x}+\underline{y}-\underline{z}$
$2\underline{k}+9$	5	00000...00000	1010	0...01110	$2\underline{x}+\underline{y}-\underline{z}$
$2\underline{k}+10$	5	00010...00000	0000	0...00101	$2\underline{x}+\underline{y}-\underline{z}$
$2\underline{k}+11$	6	00000...00000	0001	0...00000	$2\underline{x}+\underline{y}-\underline{z}$

Appendix C. Multiplication.

To program the computer for multiplication we make use of all of the "jump" instructions listed in Fig. 2. For simplicity let us suppose that the two numbers to be multiplied are positive and have been scaled into the range of the machine, so $0 \leqq \underline{x} < 1$ and $0 \leqq \underline{y} < 1$. Since $\underline{x}_0$ is 0, it is obvious that

$$\underline{x} = \underline{x}_{15}2^{-15} + \underline{x}_{14}2^{-14} + \ldots + \underline{x}_1 2^{-1}$$

and that

$$\underline{xy} = \underline{x}_{15}\underline{y}2^{-15} + \underline{x}_{14}\underline{y}2^{-14} + \ldots + \underline{x}_1\underline{y}2^{-1}$$

which is equivalent by successive partial factorings of 2^{-1} to

$$\underline{xy} = (\ldots(\underline{x}_{15}\underline{y}2^{-1} + \underline{x}_{14}\underline{y})2^{-1} + \ldots + \underline{x}_1\underline{y})2^{-1}.$$

It is convenient to define the product of two such numbers $\underline{x}$ and $\underline{y}$ by recursion, letting $\underline{P}_0 = 0$ be the <u>0th partial product</u> and $\underline{P}_i = (\underline{P}_{i-1} + \underline{x}_{16-i}\underline{y})2^{-1}$ the <u>ith partial product</u>. The product of $\underline{x}$ and $\underline{y}$ is the 15th partial product $\underline{P}_{15}$. (Because of the limited number of cells in our ARITHMETIC UNIT the product $\underline{xy}$ formed by the machine will be truncated after its 15th binary place, and equal to the correct product only within 2^{-15}.)

We program the multiplication of $\underline{x}$ and $\underline{y}$ by having the machine form the successive partial products $\underline{P}_1$, $\underline{P}_2$, ..., and stop when it has formed $\underline{P}_{15}$. Given $\underline{P}_{i-1}$ we form $\underline{P}_i$ by adding $\underline{x}_{16-i}\underline{y}$ to $\underline{P}_{i-1}$ and halving their sum. Now $\underline{x}_{16-i}\underline{y} = \underline{y}$ if $\underline{x}_{16-i} = 1$, and $\underline{x}_{16-i}\underline{y} = 0$ if $\underline{x}_{16-i} = 0$. The computer can determine whether $\underline{x}_{16-i}$ is 1 or 0 by transferring $\underline{x}2^{1-i}$ to the ARITHMETIC UNIT and sensing its rightmost digit $\underline{x}_{16-i}$, which occupies cell $\underline{a}_{15}$. If $\underline{a}_{15}$ is 1, $\underline{y}$ must be added to $\underline{P}_{i-1}$, and their sum halved to obtain $\underline{P}_i$, whereas if $\underline{a}_{15}$ is 0, $\underline{P}_i$ is obtained simply by halving $\underline{P}_{i-1}$.

To form successive partial products until $\underline{P}_{15}$ is obtained the machine must keep count of the number of partial products it has formed. This count is kept by placing an index $\underline{i}$ (initially 0) in bin 104 and adding 1 to it* as each new partial product is formed, so the index in bin 104 will be $\underline{i}$ when the machine has formed $\underline{P}_i$. If the number 15 is subtracted from $\underline{i}$, the result will be minus if $\underline{P}_{15}$ has not yet been formed (in which case we want the machine to continue the process of forming successive partial products), whereas that result will fail to be minus for the first time when $\underline{P}_{15}$ has been formed (in which case we want the machine to stop). Consequently, when the number $\underline{i}$-15 has been formed in the ARITHMETIC UNIT our next instruction should be OBEY 0 IF MINUS, and the instruction after that one should be STOP.

*See, however, the discussion in Section 6.1.

Our complete routine for the multiplication of two positive numbers $\underline{x}$ and $\underline{y}$ can now be set down as follows.

```
bin  0:  TRANSFER 101
bin  1:  OBEY 6 IF a15 IS 1
bin  2:  HALVE
bin  3:  STORE IN 101
bin  4:  TRANSFER 103
bin  5:  OBEY 10

bin  6:  HALVE
bin  7:  STORE IN 101
bin  8:  TRANSFER 102
bin  9:  ADD 103
bin 10:  HALVE
bin 11:  STORE IN 103

bin 12:  TRANSFER 104
bin 13:  ADD 105
bin 14:  SUBTRACT 106
bin 15:  OBEY 0 IF MINUS

bin 16:  STOP
  .
  .
  .
```

bin 101: $\underline{x}2^{1-i}(\underline{x}, \underline{x}2^{-1}, \ldots, \underline{x}2^{-14})$

bin 102: $\underline{y}$

bin 103: $\underline{P}_i$ $(0, \ldots, \underline{P}_{15})$

bin 104: $\underline{i}$ $(0, \ldots, 15)$

bin 105: 1

bin 106: 15.

Appendix D. Arithmetic Operations on Instruction Words.

As remarked in Section 6.1, it is sometimes convenient to have arithmetic operations performed on instruction words as well as on number words. Such operations occur in the following routine, which also illustrates the use of the instruction SUBSTITUTE IN X. The problem is to compute the sum of a collection of numbers $\underline{a}_{100}, \underline{a}_{101}, \dots, \underline{a}_{199}$ stored in bins 100, 101, ..., 199 of the parallel storage.

```
bin  0:  TRANSFER 202
bin  1:  ADD 200
bin  2:  STORE IN 202

bin  3:  SUBSTITUTE IN 4
bin  4:  TRANSFER 0
bin  5:  ADD 203
bin  6:  STORE IN 203

bin  7:  TRANSFER 202
bin  8:  SUBTRACT 201
bin  9:  OBEY 0 IF MINUS

bin 10:  STOP
  .
  .
  .
```

bin 100:	$\underline{a}_{100}$
bin 101:	$\underline{a}_{101}$
.	
.	
.	
bin 199:	$\underline{a}_{199}$
bin 200:	1
bin 201:	199
bin 202:	$\underline{i}$ (99, ..., 199)
bin 203:	$\sum_{j=100}^{\underline{i}} \underline{a}_j$ $(0, \ldots, \sum_{j=100}^{199} \underline{a}_j)$.

Appendix E. Logical Design of the ARITHMETIC UNIT.

The discussion of the ARITHMETIC UNIT in Section 5.4 conveys no idea of the process of discovering or inventing a net to realize the sixteen defining formulas there presented. In this appendix we shall attempt to show how symbolic logic can be used to achieve that end. The present discussion is intended to serve three purposes: first, to provide an alternative method of showing that the ARITHMETIC UNIT does perform the functions required of it; second, to show how symbolic logic can be used in logical design; and third, to help the reader understand more clearly how the ARITHMETIC UNIT works.

It is obvious that (K'), the sixteenth defining formula of Section 5.4, which is $\underline{a}_i(\mathcal{T}+1) \equiv \underline{r}_i(\mathcal{T})$, can be realized by sixteen delay elements with input wires $\underline{r}_i$ and output wires $\underline{a}_i$, and these will constitute the core of our ARITHMETIC UNIT.

The task of constructing the remainder of the ARITHMETIC UNIT can be viewed in the following way. We have sixteen output wires $\underline{r}_i$ whose states are to be determined in every case by the states of all the input wires $\underline{t}$, $\underline{a}$, and the five control wires $\underline{A}_r$, $\underline{A}_h$, $\underline{A}_c$, $\underline{A}_{ls}$, $\underline{A}_{rs}$. Our task is to construct a net from these inputs to outputs $\underline{r}_i$ which will be governed by the fifteen remaining defining formulas of Section 5.4. That task is easily accomplished if we can find an expression of the form

$$\underline{r}_i \equiv \underline{F}(\underline{t}, \underline{a}, \underline{A}_r, \underline{A}_h, \underline{A}_c, \underline{A}_{ls}, \underline{A}_{rs})$$

for each $\underline{r}_i$, such that these expressions logically entail the remaining fifteen defining formulas. For any net which realizes these expressions will also realize any formulas logically entailed by them. And once the $\underline{r}_i$'s are expressed as explicit functions of the other terms, we can construct the net quite mechanically from the elements explained in Fig. 3.

In carrying through the program sketched in the preceding paragraph we shall find it convenient (though it is not necessary) to consider wires $\underline{r}_i$ and $\underline{c}_{i-1}$ as outputs whose states are determined by the states of the input wires $\underline{c}_i$, $\underline{t}_i$, $\underline{a}_{i-1}$, $\underline{a}_i$, $\underline{a}_{i+1}$, and the five control wires. The fifteen defining formulas of Section 5.4 describe the ways in which the states of these output wires are determined by the states of those input wires. But those fifteen conditional formulas express the functional dependence of $\underline{r}_i$ and $\underline{c}_{i-1}$ on the other terms not explicitly but only implicitly. What we now wish to find is an expression of the form

$$(E_r) \qquad \underline{r}_i \equiv \underline{F}(\underline{c}_i, \underline{t}_i, \underline{a}_{i-1}, \underline{a}_i, \underline{a}_{i+1}, \underline{A}_r, \underline{A}_h, \underline{A}_c, \underline{A}_{ls}, \underline{A}_{rs})$$

for each $\underline{r}_i$, and an expression of the form

$$(E_c) \qquad \underline{c}_{i-1} \equiv \underline{F}(\underline{c}_i, \underline{t}_i, \underline{a}_{i-1}, \underline{a}_i, \underline{a}_{i+1}, \underline{A}_r, \underline{A}_h, \underline{A}_c, \underline{A}_{ls}, \underline{A}_{rs})$$

for each c_{i-1}, in which r_i and c_{i-1} appear as explicit functions of the other terms. Of course we must also find an expression of the form

$$(E_{15}) \qquad c_{15} \equiv F(A_r, A_h, A_c, A_{1s}, A_{rs})$$

to complete our program.

The defining formulas governing r_i and c_{i-1} are all of the form

$$(I_r) \qquad G(A_r,A_h,A_c,A_{1s},A_{rs}) \supset \{r_i \equiv H(c_i,t_i,a_{i-1},a_i,a_{i+1})\},$$

$$(I_c) \qquad G(A_r,A_h,A_c,A_{1s},A_{rs}) \supset \{c_{i-1} \equiv H(c_i,t_i,a_{i-1},a_i,a_{i+1})\},$$

and

$$(I_{15}) \qquad G(A_r,A_h,A_c,A_{1s},A_{rs}) \supset \{c_{15} \equiv H(c_i,t_i,a_{i-1},a_i,a_{i+1})\},$$

in which r_i and c_{i-1} appear as implicit rather than explicit functions of the other terms. From them, however, it is easy to derive the desired expressions of the type (E_r) and (E_c) by means of the following theorem, in whose statement we use the notation $\sum_{j=1}^{J} \alpha_j$ for $\alpha_1 \vee \alpha_2 \vee \ldots \vee \alpha_J$ and $\prod_{j=1}^{J} \alpha_j$ for $\alpha_1 \cdot \alpha_2 \cdot \ldots \cdot \alpha_J$.

Theorem _J_. If $\sum_{j=1}^{J} \alpha_j$ and $\alpha_i \supset \overline{\alpha_j}$ for every $i \neq j$, $\prod_{j=1}^{J} \{\alpha_j \supset (\beta \equiv \gamma_j)\}$ if and only if $\beta \equiv \sum_{j=1}^{J} \alpha_j \gamma_j$.

Since the antecedents of the various (I_r) formulas cover all possible cases, exactly one of them being true at any given time that the ARITHMETIC UNIT is to function, Theorem J can be applied to obtain the desired expression

$$[E_r] \qquad \underline{r}_i \equiv \{(\underline{A}_r\underline{A}_h\overline{\underline{A}}_c)(\underline{t}_i \not\equiv \underline{a}_i \not\equiv \underline{c}_i) \vee (\underline{A}_r\underline{A}_h\underline{A}_c)(\overline{\underline{t}}_i \not\equiv \underline{a}_i \not\equiv \underline{c}_i) \vee \ldots \vee \overline{\underline{A}}_r\underline{a}_i\}.$$

In the same way Theorem J applies to (I_c) to yield the desired expressions

$$[E_c] \qquad \underline{c}_{i-1} \equiv \{(\underline{A}_r\underline{A}_h\overline{\underline{A}}_c)(\underline{t}_i\underline{a}_i \vee \underline{t}_i\underline{c}_i \vee \underline{a}_i\underline{c}_i) \vee (\underline{A}_r\underline{A}_h\underline{A}_c)(\overline{\underline{t}}_i\underline{a}_i \vee \overline{\underline{t}}_i\underline{c}_i \vee \underline{a}_i\underline{c}_i) \vee (\underline{A}_r\overline{\underline{A}}_h\underline{A}_c)(\underline{t}_i\underline{c}_i)\},$$

for $\underline{i} > 0$, and

$$[E_{15}] \qquad \underline{c}_{15} \equiv \{\underline{A}_r\underline{A}_c\}.$$

This last expression is equivalent to

$$[E'_{15}] \qquad \underline{c}_{15} \equiv \underline{A}_c$$

since inspection of Fig. 2 shows that $\underline{A}_c \equiv (\underline{A}_r\underline{A}_c)$.

From these expressions we can pass directly to a net constructed out of the elements of Fig. 3. But such a net, while realizing the defining formulas, would not do so very economically. It would have, for example, different 3-input inequivalence elements for the first two disjuncts of $[E_r]$, whereas it is possible to use just one such inequivalence element, as in Fig. 7.

To construct a more minimal net we first alter some of the defining formulas of Section 5.4 to bring them into the same general form as the others. The formulas whose labels are (A_1^m), (A_2^m), (A_3^m), (S_1^m), (S_2^m), (S_3^m) remain unchanged.

But (T_1^m) is replaced by three formulas, by the following considerations. In Section 5.4 instruction 6 was regarded simply as demanding the replacement of $\underline{r}$ by $\underline{t}$. We can equally well regard it as demanding the replacement of $\underline{r}$ by $\underline{t}+0$, which makes it a special case of addition. By substituting $\underline{r}_i$ for $\underline{z}_i$, $\underline{t}_i$ for $\underline{x}_i$, 0 for $\underline{y}_i$, and $\underline{c}_i$ for ξ_i in the (A) equations of Section 5.1, we obtain in place of the single formula (T_1^m) the three formulas

$[T_1^m]$ $\quad \underline{A}_r\overline{\underline{A}}_h\overline{\underline{A}}_c \supset \{\underline{r}_i \equiv (\underline{t}_i \not\equiv 0 \not\equiv \underline{c}_i)\}$,

$[T_2^m]$ $\quad \underline{A}_r\overline{\underline{A}}_h\overline{\underline{A}}_c \supset \{\underline{c}_{i-1} \equiv (\underline{t}_i 0 \vee \underline{t}_i\underline{c}_i \vee 0\underline{c}_i)\}$ for $\underline{i} > 0$,

$[T_3^m]$ $\quad \underline{A}_r\overline{\underline{A}}_h\overline{\underline{A}}_c \supset \{\underline{c}_{15} = 0\}$,

to be realized by the ARITHMETIC UNIT.

Similarly, the three formulas for TRANSFER COMPLEMENT X are written as

$[TC_1^m]$ $\quad \underline{A}_r\overline{\underline{A}}_h\underline{A}_c \supset \{\underline{r}_i \equiv (\overline{\underline{t}}_i \not\equiv 0 \not\equiv \underline{c}_i)\}$,

$[TC_2^m]$ $\quad \underline{A}_r\overline{\underline{A}}_h\underline{A}_c \supset \{\underline{c}_{i-1} \equiv (\overline{\underline{t}}_i 0 \vee \overline{\underline{t}}_i\underline{c}_i \vee 0\underline{c}_i)\}$ for $\underline{i} > 0$,

$[TC_3^m]$ $\quad \underline{A}_r\overline{\underline{A}}_h\underline{A}_c \supset \{\underline{c}_{15} = 1\}$.

Just as the transfer can be treated as machine addition to zero, so doubling can be treated as the combination of a left shift (to accomplish the doubling) with an addition of zero. Since $\underline{t} = 0$ whenever instruction 8 is being executed, doubling can be treated as the combination of left shift with addition of $\underline{t}$. This treatment permits us to develop, for instruction 8, formulas of the same general type as for the others. Here we obtain four formulas

$[D_1^m]$ $\quad \underline{A}_r\underline{A}_{ls} \supset \{\underline{r}_i \equiv (\underline{t}_i \not\equiv \underline{a}_{i+1} \not\equiv \underline{c}_i)\}$ for $\underline{i} < 15$,

$[D_2^m]$ $\quad \underline{A}_r\underline{A}_{ls} \supset \{\underline{c}_{i-1} \equiv (\underline{t}_i\underline{a}_{i+1} \vee \underline{t}_i\underline{c}_i \vee a_{i+1}\underline{c}_i)\}$ for $\underline{i} > 0$,

$[D_3^m]$ $\quad \underline{A}_r\underline{A}_{ls} \supset \{\underline{c}_{15} = 0\}$,

$[D_4^m]$ $\quad \underline{A}_r\underline{A}_{ls} \supset \{\underline{r}_{15} = 0\}$.

By a parallel route we arrive at four formulas to be realized by the ARITHMETIC UNIT in executing instruction 9.

$[H_1^m]$ $\quad \underline{A}_r\underline{A}_{rs} \supset \{\underline{r}_i \equiv (\underline{t}_i \not\equiv \underline{a}_{i-1} \not\equiv \underline{c}_i)\}$ for $\underline{i} > 0$,

$[H_2^m]$ $\quad \underline{A}_r\underline{A}_{rs} \supset \{\underline{c}_{i-1} \equiv (\underline{t}_i\underline{a}_{i-1} \vee \underline{t}_i\underline{c}_i \vee \underline{a}_{i-1}\underline{c}_i)\}$ for $\underline{i} > 0$,

$[H_3^m]$ $\quad \underline{A}_r\underline{A}_{rs} \supset \{\underline{c}_{15} = 0\}$,

$[H_4^m]$ $\quad A_rA_{rs} \supset \{\underline{r}_0 = \underline{a}_0\}$.

These twenty formulas take the place of the first fourteen defining formulas of Section 5.4. We construct a more minimal net by first deriving a smaller number of expressions which will logically entail these twenty. To exploit the fact that all of the twenty which have the same subscript have the same general form, and that $\underline{A}_r$ occurs in every one of their antecedents, we introduce the auxiliary variables $\underline{t}'_i$, $\underline{w}_i$, and ε, and construct an expression of the form

$[E_1]$ $\quad \underline{A}_r \supset \{\underline{r}_i \equiv (\underline{t}'_i \not\equiv \underline{w}_i \not\equiv \underline{c}_i)\}$

which will entail (A_1^m), (S_1^m), $[T_1^m]$, $[TC_1^m]$, $[D_1^m]$, and $[H_1^m]$. We also try to construct an expression

$[E_2]$ $\quad \underline{A}_r \supset \{\underline{c}_{i-1} \equiv (\underline{t}'_i\underline{w}_i \vee \underline{t}'_i\underline{c}_i \vee \underline{w}_i\underline{c}_i)\}$

which will entail (A_2^m), (S_2^m), $[T_2^m]$, $[TC_2^m]$, $[D_2^m]$, and $[H_2^m]$, and an expression

[E3] $\quad \underline{A}_r \supset \{\underline{c}_{15} \equiv \mathcal{E}\}$

which will entail (A_3^m), (S_3^m), $[T_3^m]$, $[TC_3^m]$, $[D_3^m]$, and $[H_3^m]$. When they have been constructed they can be readily combined with $[D_4^m]$, $[H_4^m]$, and (K) to give us a set of expressions whose realization by the ARITHMETIC UNIT will enable it to fulfill its part in the execution of all arithmetic instructions.

Let us first consider $\underline{t}_i'$. By inspection we note that when $\underline{A}_c$ is stimulated ($[S^m]$, $[TC^m]$), $\underline{t}_i'$ is $\bar{\underline{t}}_i$, whereas in all other cases ($[A^m]$, $[T^m]$, $[D^m]$, $[H^m]$), $\underline{t}_i'$ is $\underline{t}_i$. Hence we have

$$\underline{A}_c \supset (\underline{t}_i' \equiv \bar{\underline{t}}_i) \quad \text{and} \quad \bar{\underline{A}}_c \supset (\underline{t}_i' \equiv \underline{t}_i),$$

and by Theorem J

$$\underline{t}_i' \equiv (\underline{A}_c\bar{\underline{t}}_i \vee \bar{\underline{A}}_c\underline{t}_i)$$

or

[L°] $\quad \underline{t}_i' \equiv (\underline{A}_c \not\equiv \underline{t}_i).$

Next let us consider $\underline{w}_i$. When $\underline{A}_h$ is stimulated ($[A^m]$, $[S^m]$), $\underline{w}_i$ is $\underline{a}_i$; when $\underline{A}_{ls}$ is stimulated ($[D^m]$), $\underline{w}_i$ is $\underline{a}_{i+1}$; when $\underline{A}_{rs}$ is stimulated ($[H^m]$), $\underline{w}_i$ is $\underline{a}_{i-1}$; and in all other cases ($[T^m]$, $[TC^m]$), $\underline{w}_i$ is 0. Hence we obtain

[M°] $\quad \underline{w}_i \equiv (\underline{A}_h\underline{a}_i \vee \underline{A}_{ls}\underline{a}_{i+1} \vee \underline{A}_{rs}\underline{a}_{i-1})$ for $0 < \underline{i} < 15$.

By $[D_4^m]$, for $\underline{w}_{15}$ we drop the second disjunct of [M°]; and by $[H_4^m]$, for $\underline{w}_0$ we replace the third disjunct of [M°]

by $A_{rs}a_0$. Here we may replace the restriction on i in $[M°]$ by the convention that

$$a_{16} = 0 \quad \text{and} \quad a_{-1} = a_0.$$

Finally let us consider those of the twenty defining formulas whose labels have a subscript 3. When A_c is not stimulated ($[A_3^m]$, $[T_3^m]$, $[D_3^m]$, $[H_3^m]$), $c_{15} = 0$. When A_c is stimulated ($[S_3^m]$, $[TC_3^m]$), $c_{15} = 1$. Hence $[E_3]$ becomes

$[E_3']$ $\quad A_r \supset (c_{15} \equiv A_c)$.

We now assert, without proof, that formulas $[E_1]$, $[E_2]$, $[L°]$, $[M°]$, and $[E_3']$ jointly entail all twenty of our defining formulas.*

---

*The proof is complicated, but may be facilitated by using appropriate versions of the following theorem, in whose statement we use the notation $S_\alpha^\beta F$ for the result of replacing every occurrence of the variable β in F by the expression α :

Theorem M. $\alpha \supset S^\gamma_{\beta \not\equiv \gamma} F$ if and only if both $\alpha\beta \supset F$ and $\alpha\bar{\beta} \supset S^\gamma_{\bar{\gamma}} F$.

An example of its use is the following. Inserting in formula $[E_1]$ the value of t_i' given by $[L°]$, we obtain the expression

$$A_r \supset \{r_i \equiv [(A_c \not\equiv t_i) \not\equiv w_i \not\equiv c_i]\} .$$

Now applying Theorem M to that expression we obtain

$$A_rA_c \supset \{r_i \equiv (t_i \not\equiv w_i \not\equiv c_i)\}$$

and

$$A_r\bar{A}_c \supset \{r_i \equiv (\bar{t}_i \not\equiv w_i \not\equiv c_i)\} .$$

(footnote continued p. 76)

We can now use Theorem J to derive an expression of the desired form (E_r) which is equivalent to (K) and $[E_1]$. The derived expression is

$$[N] \qquad \underline{r}_i \equiv [\overline{\underline{A}}_r\underline{a}_i \vee \underline{A}_r(\underline{t}'_i \not\equiv \underline{w}_i \not\equiv \underline{c}_i)] \; .$$

We can also derive an expression of the desired form (E_c) from $[E_2]$ and the formula

$$\overline{\underline{A}}_r \supset \{\underline{c}_{i-1} \equiv (\underline{t}'_i\underline{w}_i \vee \underline{t}'_i\underline{c}_i \vee \underline{w}_i\underline{c}_i)\}$$

which can be stipulated to hold since (K) places no restriction on $\underline{c}_{i-1}$. The derived expression is

$$[P^\circ] \qquad \underline{c}_{i-1} \equiv (\underline{t}'_i\underline{w}_i \vee \underline{t}'_i\underline{c}_i \vee \underline{w}_i\underline{c}_i) .$$

(Footnote from p. 75 continued)
In these two formulas we insert the value of $\underline{w}_i$ given by $[M^\circ]$ to obtain

$$\underline{A}_r\underline{A}_c \supset \{\underline{r}_i \equiv [\underline{t}_i \not\equiv (\underline{A}_h\underline{a}_i \vee \underline{A}_{ls}\underline{a}_{i+1} \vee \underline{A}_{rs}\underline{a}_{i-1}) \not\equiv \underline{c}_i]\}$$

and

$$\underline{A}_r\overline{\underline{A}}_c \supset \{\underline{r}_i \equiv [\overline{\underline{t}}_i \not\equiv (\underline{A}_h\underline{a}_i \vee \underline{A}_{ls}\underline{a}_{i+1} \vee \underline{A}_{rs}\underline{a}_{i-1}) \not\equiv \underline{c}_i]\} \; .$$

Since $\underline{A}_h$ $(\underline{A}_h = 1)$ implies both $\underline{A}_{rs} = 0$ and $\underline{A}_{ls} = 0$, from the two formulas above we obtain

$$\underline{A}_r\underline{A}_h\overline{\underline{A}}_c \supset \{\underline{r}_i \equiv (\underline{t}_i \not\equiv \underline{a}_i \not\equiv \underline{c}_i)\}$$

and

$$\underline{A}_r\underline{A}_h\underline{A}_c \supset \{\underline{r}_i \equiv (\overline{\underline{t}}_i \not\equiv \underline{a}_i \not\equiv \underline{c}_i)\} \; ,$$

which are (A^m_1) and (S^m_1) of our defining formulas. The other defining formulas may be derived from $[E_1]$, $[E_2]$, $[L^\circ]$, $[M^\circ]$, and $[E'_3]$ by similar procedures.

Thus [P°] implies [E_2] , though not conversely. The net realizing [P°] is satisfactory, however, since any net which realizes [P°] will also realize [E_2].

Now we introduce a new variable, $\underline{u}_i$, which we define by means of the equivalence

[Q°] $\underline{u}_i \equiv (\underline{t}'_i \not\equiv \underline{w}_i \not\equiv \underline{c}_i)$.

If we now replace the expression which defines $\underline{u}_i$ by $\underline{u}_i$ in [N], we obtain

[N°] $\underline{r}_i \equiv (\underline{\bar{A}}_r\underline{a}_i \vee \underline{A}_r\underline{u}_i)$.

The degree expressions [L°] , [M°] , [N°] , [P°] , [Q°] constitute the definition of the ARITHMETIC UNIT which was sought. Any net which realizes them will perform the functions required, and it is easy to construct a net which realizes them out of the elements explained in Fig. 3. This net is in fact Fig. 7 (less the elements used in accomplishing the ARITHMETIC UNIT's transmission function).