THE BELL SYSTEM
TECHNICAL JOURNAL

VOLUME XXXVII NOVEMBER 1958 NUMBER 6

Functional Design of a Stored-Program Electronic Switching System

By HOWARD N. SECKLER and JOHN J. YOSTPILLE

(Manuscript received July 30, 1958)

The design of an experimental electronic switching system presented many new problems in specifying the functions to be performed and in developing the means to carry them out. This resulted in the use of a number of techniques new to the telephone switching field. This paper, divided into two parts, describes some of these. Part One deals with a specification of what the system must do and with its general mode of operation; Part Two describes a control philosophy employing a stored program.

PART ONE

I. INTRODUCTION

In the development of a large system, one of the most important problems is completely specifying its internal organization. The task is a complex one in the case of a telephone switching system because of the multiplicity of logical interactions involved. This paper discusses this problem as applied to an experimental electronic telephone switching system developed by Bell Telephone Laboratories.*

During the course of designing the system many techniques new to the telephone field were employed. These include a stored-program con-

* The system was described in a paper[1] in the previous issue which provides pertinent background material for this article.

trol philosophy and separation of the basic system functions, such as memory and logic, into efficient functionally concentrated blocks of equipment.

The experimental system, a prototype of a practical telephone office, is a real-time special-purpose machine. One of the ever-present problems was that of effecting a balance between real-time occupancy, amount of memory and other equipment, and complexity of system operation. In this balance good telephone service and over-all economy are the ultimate objectives.

Finally, a method of system operation had to be developed in order to process numerous types of telephone calls. An ideal solution is one where there is no interdependence between equipment requirements and telephone call functions. This would give rise to a universal system in which the addition or deletion of features or traffic in a telephone office would require no basic equipment changes. Within limits, this ideal solution was approached in the experimental system.

II. BASIC SYSTEM ORGANIZATION

The experimental system is composed of six major components: the switching network, the scanner, the barrier grid store, the flying spot store, the signal distributor and the central control. They are shown in Fig. 1 with their functional interconnections.

The function of the switching network[2] is to provide voice transmission connections between telephone customers; it can be considered as the primary output of the system. The switching network is made up of two main parts: the distribution switching network and the concentrators. Each customer line terminates on a concentrator, which in turn has access to the distribution switching network. Connections are established in the switching network by means of orders given sequentially to the two network controls, the concentrator marker and the distribution marker. The functions assigned to the switching network are kept as simple as possible to minimize the amount of network control equipment required. Network actions are requested by one of three orders (connect, release or trace), accompanied by appropriate terminal addresses. The network markers report the outcome of each order.

The scanner[3] is an input circuit. It is capable, on a one-at-a-time basis, of determining the state ("on-hook" or "off-hook") of each line and trunk in the office and is used to gather supervisory and dialed information. A particular customer line is interrogated by directing the scanner to a specific address; the scanner, in turn, gives an answer corresponding to the state of the line.

The barrier grid store[4] provides bulk temporary (erasable) memory for the system. It is used to store dynamic records such as line and trunk busy-idle states and dialed information. This information is stored in binary form and is read one bit at a time with random address access. A particular bit can be read and rewritten by appropriately addressing the store and giving one of four possible orders: read and regenerate, read and change, read and write zero or read and write one.

The flying spot store[5] is a bulk semipermanent memory file. It holds the telephone translation records and the operational program, and information is read out of the store on a word-organized basis. There are two general types of orders that may be given to the flying spot store: read the next program order according to a predetermined sequence or read an order at any specified location. These are known as the "advance" and "transfer" orders, respectively.

The signal distributor[6] is an output circuit that is capable of operating any one of a number of flip-flops which in turn may operate an electromechanical relay. Its prime use is for controlling relays in trunk circuits that communicate with conventional electromechanical systems. It provides compatibility between the high-speed electronic system and lower-speed mechanical systems.

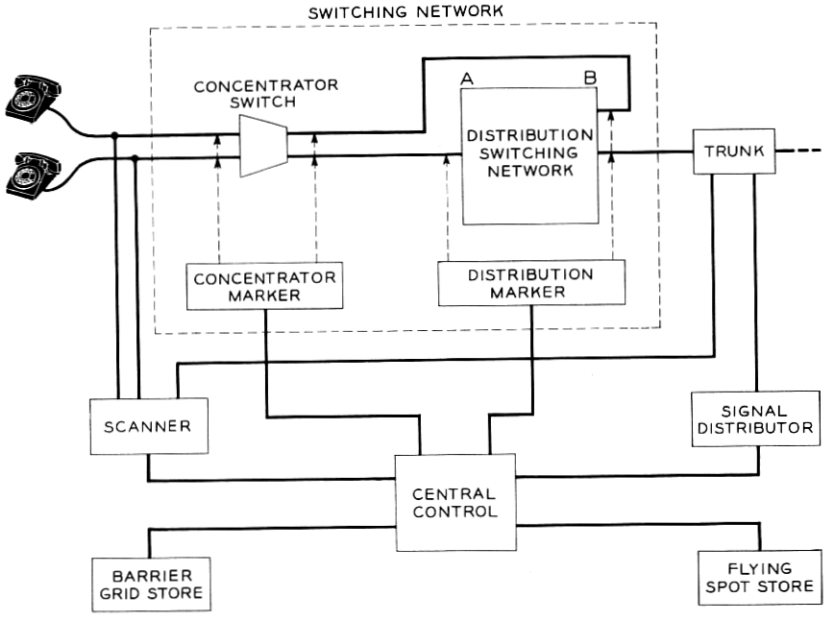


Fig. 1 — Block diagram of experimental electronic switching system.

The central control logic and organization is described in detail in Part Two of this paper. It is a logic circuit which responds to the operational program stored in the flying spot store. It controls all of the major system components, makes decisions (usually binary) based on answers it receives from them and selects the next program order to be executed.

III. SPECIFICATION OF SYSTEM SEQUENCES

Telephone switching systems are required to perform a large number of complex telephone call and administrative functions. It is necessary to specify these functions and the method of accomplishing them in exacting detail to integrate them into the system and to expedite the

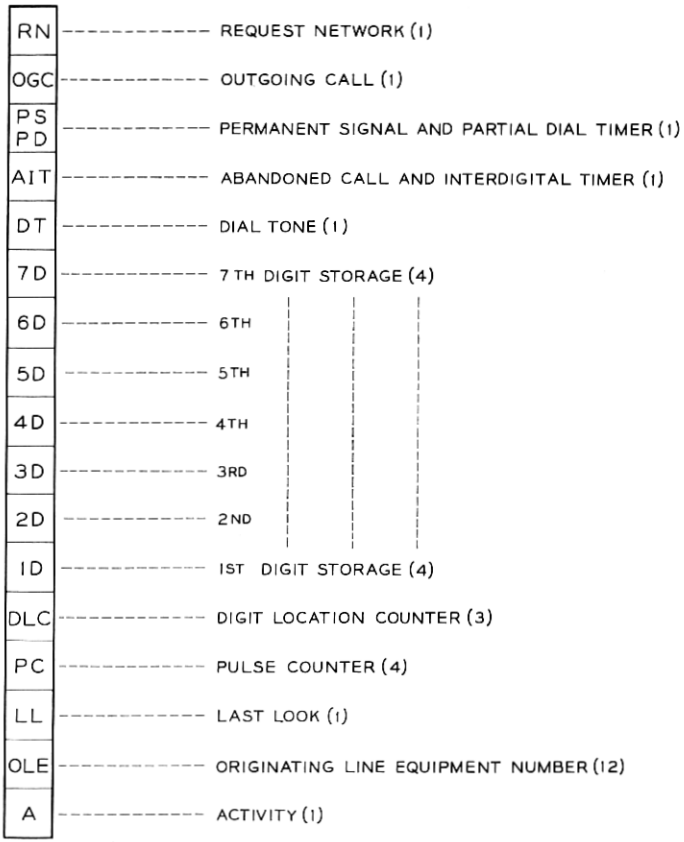


Fig. 2 — Originating register layout.

preparation of a (stored) program. Conceptually, the program itself could be written directly from broad requirements. However, it was found that a link was needed between the required external features and the specific means for implementing them in the experimental system. This link was provided in the form of detailed "sequence charts."

Before system sequence charts can be written, a language must be established. Since the charts describe the operation of a specific system, their form is somewhat tailored to the characteristics of that system. In the design of the experimental system sequences, the operators Read, Write, Transport and Add, abbreviated R, W, T and A respectively, were found to be sufficient to satisfy a great majority of the requirements. These operators generally act on some specified memory element or system component and, in combination with the identity of the memory element or system component, form a sequence-chart "order." A typical sequence-chart order might be: *Read the output of the scanner*, abbreviated R(S).

It is convenient to refer to locations, units of equipment and other items in abbreviated form on sequence charts. As described in the previous issue[1], barrier grid store memory is allocated in the form of "registers" performing specific functions. For example, Fig. 2 shows the layout of an originating register, which provides the temporary memory required for a customer while a call is being dialed. Storage locations in the barrier grid store are generally represented by an abbreviation designating the register followed by a hyphen and the abbreviation for the spot (bit) or spots within the register. For example, OR-LL means the "last look" spot in the originating register. The prefix may be omitted when the identity of the register is obvious. Other commonly used abbreviations are:

S for scanner,

D for signal distributor,

N as a prefix for flip-flops associated with the switching network,

FF as a prefix for specified flip-flops in the central control,

CC for unspecified flip-flops in the central control.

The four basic types of orders, described in more detail are:

1. The Read order has the general form of R (X) (Y) where X is the identity of the information to be read and Y is its storage location. Either X or Y may be omitted if not needed. A decision is always made as the result of a read order.

2. The Write order has the form $W(K) \rightarrow (Z)$, where K is a specific fixed number, either in numerical or symbolic form, and Z is the storage location into which the information K is to be written.

3. The Transport order has the general form $T(X)\ (Y) \rightarrow (Z)$, where X is the identity of the information at storage location Y and Z is the storage destination of X. As above, either X or Y can be omitted if not needed. This order is used to move any information from one location to another.

4. The Add order, which is also used for subtraction, is of the form $A(K) \rightarrow Z$ or $A(-K) \rightarrow Z$, where K is a positive integer and Z a storage location.

IV. SYSTEM SEQUENCES AND TECHNIQUES

Although the experimental system is limited in scope, a large number of sequence charts are needed to specify its functional operation. A few sequences followed at the start of a telephone call are discussed in this section in order to illustrate how certain typical functions are accomplished and what techniques are used. For the sake of brevity, many details have been omitted, particularly in the latter part of this section.

4.1 *Supervisory Functions*

One of the functions of a telephone switching system is the recognition of supervisory signals from customers, that is, new requests for service and terminations of established calls. The customer signals the telephone office by lifting his telephone receiver from the switchhook (off-hook) to request service and later, by placing it back on the switchhook (on-hook), to terminate service. These actions respectively close and open a metallic path to the telephone office. Table I shows the scanner output for each state.

Since the scanner can only determine the existing state of a customer line, some memory must be provided in order to detect changes in line state. This is done by assigning two barrier grid store bits to each line. These are referred to here as the first and second line memory bits, or L1 and L2 spots. The experimental system uses three of the four possible combinations of each pair of line memory bits, with assignments as shown in Table II. The "served by a register" state is assigned when

TABLE I — LINE STATES AND CORRESPONDING SCANNER OUTPUTS

Telephone Receiver	Telephone Line	Scanner Output
on-hook	open	0
off-hook	closed	1

TABLE II — LINE MEMORY ASSIGNMENT

Status of Line	Barrier Grid Store Memory Bits	
	L1	L2
Idle	0	0
Talking	1	0
Served by a Register	0	1

special action is taken on a line such as dial pulse recording, ringing or disconnect timing.

The experimental system starts action on supervisory signals from customers within 100 milliseconds of their occurrence. Fig. 3 illustrates, in sequence chart form, the work that may be done on each customer line once every 100 milliseconds; reference to Table III will be helpful in understanding the description that follows.

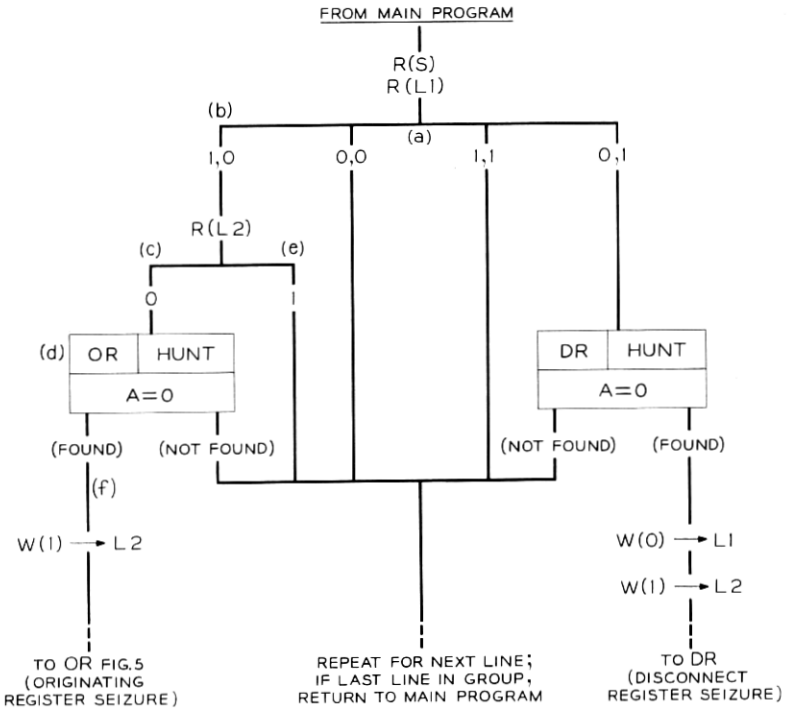


Fig. 3 — 100-millisecond supervisory line scan.

TABLE III — LINE SIGNALS RECOGNIZED IN SUPERVISORY SCAN SEQUENCE BY COMPARING SCANNER OUTPUT WITH LINE SPOTS

Scanner Output	Line Spots		Customer Signal
	L1	L2	
0	0	0	None
1	1	0	None
1	0	0	Call Originated
0	1	0	Disconnect
0 or 1	0	1	Line under control of other sequences

As described in a previous paper[1] the "main program" is the vehicle which assigns the multiplicity of real-time functions which the system must perform at various periodic intervals. At the start of the supervisory scan for originations and disconnects, the scanner is directed to the first line of a predetermined group and the barrier grid store is addressed to read its associated L1 spot; this is specified by the symbols R(S) and R(L1) near the top of the sequence chart. This leads to a branch (point *a*) in the sequence which opens to four possible paths. If the answers received from the scanner and barrier grid store match (0,0 or 1,1), no further action is taken on the line (see Table III) and the system advances to the next line in the group. However, if the combination (1,0) is returned, the L2 spot must be examined (point *b*) to determine whether it is in the zero state, indicating that a call has been originated by this line in the previous 100-millisecond interval. If this is the case (point *c*), a hunt for an idle originating register (OR) is initiated, as specified by the symbolic box in the left leg of the chart (point *d*). If the L2 spot reads 1 (point *e*), the system advances to the next line in the group, as above. Proceeding to the output of the idle register hunt box, if an idle register is found (point *f*), a 1 is recorded in the L2 spot, putting the line in the served-by-a-register state, and the system advances to a sequence for seizing the originating register. If no idle register is found, the above action will be repeated once every 100 milliseconds until a register becomes available. The leg on the right-hand side of Fig. 3 shows the action taken when a disconnect is detected. The sequence of system actions represented by the boxes of Fig. 3 is shown in Fig. 4.

Although the above example is a simple one, it illustrates certain fundamental principles. The majority of customer lines will always be in the quiescent states of idle (0,0) or talking (1,0), and only a small amount of system action is required for them once every 100 milli-

seconds. Only when a line changes state does the system have a complex job to perform. Since the number of changes of line state per 100-millisecond interval is very small, the percentage of real time consumed by the 100-millisecond line-scan sequence is correspondingly small. Another factor to be considered is the amount of memory required for each function. Here two barrier grid store (BGS) bits are assigned to each line and a certain amount of program space in the flying spot store is used for the supervisory scan function. It is apparent after some study that a decrease in the amount of BGS memory assigned to each line will result in an increase in real-time consumption and an increase in program space. To show that a good balance, in this respect, has been attained in the experimental system is beyond the scope of this article.

4.2 *Originating Register Seizure*

In the experimental system, BGS memory is associated with certain telephone functions in a manner somewhat similar to the use of common control equipment such as registers, senders and markers in conventional relay telephone switching systems. As mentioned previously, the originating register illustrated in Fig. 2 shows the BGS memory that is associated

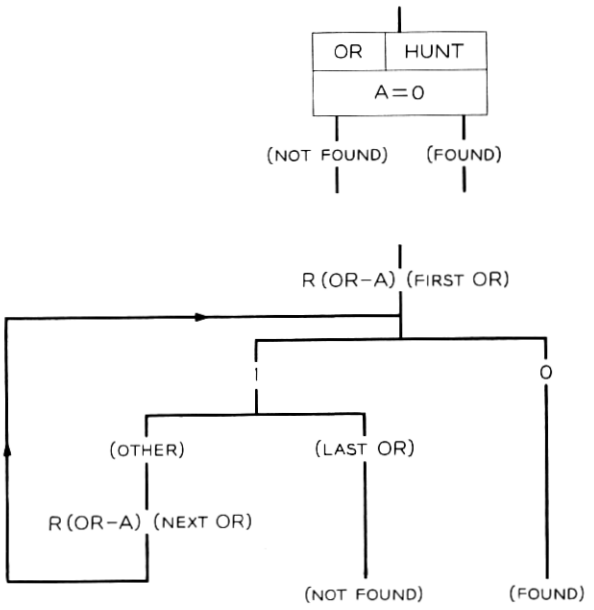


Fig. 4 — Register hunt.

with a line during dialing. For each bit or subgroup of bits, the functional abbreviation, the full name and the number of bits (in parenthesis) are given. The originating register is one type of a class of BGS registers referred to as "call registers." In this system a more than sufficient number of call registers to give a good grade of service can be economically provided, since no equipment other than the memory itself is affected by the number of registers.

After the origination of a new call has been detected and an idle originating register is found, as shown in Figs. 3 and 4, the originating register must be seized and action started to connect a dial tone trunk to the line via the concentrator and distribution network. The system sequences for carrying out this task are shown in Fig. 5. While a register is idle the only useful information recorded in it is a 0 in the "activity" (busy-idle) spot. This eliminates the need for keeping the remaining register memory bits in a fixed state over long periods of inactivity, as

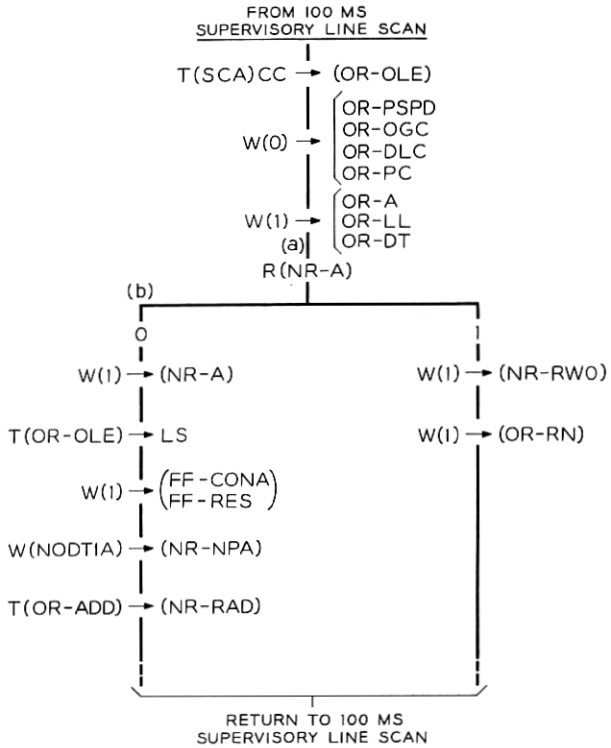


Fig. 5 — Originating register seizure.

would be the case for the last-choice registers in a large group. Therefore, when a register is first seized various state and reference information must be recorded, as shown by the Write orders in the top center leg of Fig. 5.

4.3 *Dial Tone Connection*

The switching network operates at a speed of approximately one millisecond per task, while the rest of the system performs operations at a microsecond rate. For this and other reasons buffer memory associated with the switching network is provided in the BGS and has been designated as the "network register" (NR). The network register is one type of a second class of BGS registers which are referred to as "control registers." Usually only a single control register is needed for a particular function such as network control.

After the originating register is seized, the network register activity spot (NR-A) is read as shown in Fig. 5. If a 1 is read, indicating that the network is busy, a request for later action is recorded in both the originating and network registers. If a 0 is read, indicating that the network is available, the network register is seized. In this case, the line equipment number (OR-OLE) is passed to the network line selector (LS) in the concentrator marker and certain control flip-flops are set to start connecting the line through the concentrator to a distribution network terminal. The type of action required (NODT1A—first part of dial tone connection) and address of the originating register (OR-ADD) are also recorded in the network register so the system will know where to continue in the program when this segment of the network action is completed. Having done this, the system returns to sample the condition of the next line in the 100-millisecond supervisory line scan.

Periodically the system checks the switching network to determine if it has completed a task. This is done by reading a "break in" flip-flop (FF-NBI) associated with the network control circuits, as shown in Fig. 6. When the switching network has completed an assignment, its own control circuit sets FF-NBI so that a 1 will be read the next time it is interrogated. This in turn will cause a break in the main program and will direct the system to determine, from information previously stored in the network register, what task the network has completed and then, from answers received from network control circuits, what action should be taken next. The remainder of the dial tone connection is established in this way.

The "simple-task-at-a-time" method of operating the network permits efficient utilization of the common system memory and central control, thus minimizing the quantity of equipment individual to the network. It also provides a means of coupling the lower-speed network to the rest of the system.

4.4 *Detection of Dial Pulses*

After a line has been assigned to an originating register and a dial tone connection is established, means must be provided for detecting and storing dial pulses. This is accomplished by the sequence shown in Fig. 7. The main program directs the system to carry out this sequence for all active originating registers once every 10 milliseconds, as specified by the symbolic box shown at the top of the figure. The pulses generated by the customer's dial are a series of momentary "on-hook" intervals, the number corresponding to the digit dialed, except for zero which is represented by 10 pulses. The 10-millisecond scanning rate is fast enough to insure that every legitimate dial pulse and the "off hook" intervals that separate them will be sampled at least once[3].

After the originating line equipment number (OLE) recorded upon seizure of the originating register has been read as indicated in Fig. 7, the scanner is directed to scan the line and the BGS is addressed to read the last look spot (LL). This spot contains an up-to-date record of the state of the line according to the last previous scanner reading (0 for "on-hook" and 1 for "off-hook"). If the answers received from the scanner and BGS match (0,0 or 1,1) no change has taken place since

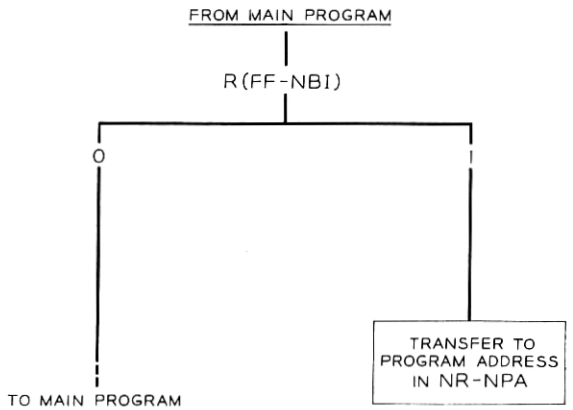


Fig. 6 — Periodic network check.

the last scan and the program proceeds to the next originating register. However, if the scanner reads 0 and the LL spot 1, this is interpreted as the leading edge of a dial pulse; in this case, the LL spot is put up to date (0) and the pulse counter spots (PC) are incremented by one. In addition, the abandoned call and interdigital timer spot (AIT) is written to 0, thus permitting another sequence (not shown) to detect an abandoned call if the on-hook condition persists. The dial tone spot (DT) is also read to determine if dial tone is connected to the customer line. Dial tone is disconnected from the line as soon as the first pulse of the first dialed digit is received, by a method similar to the one

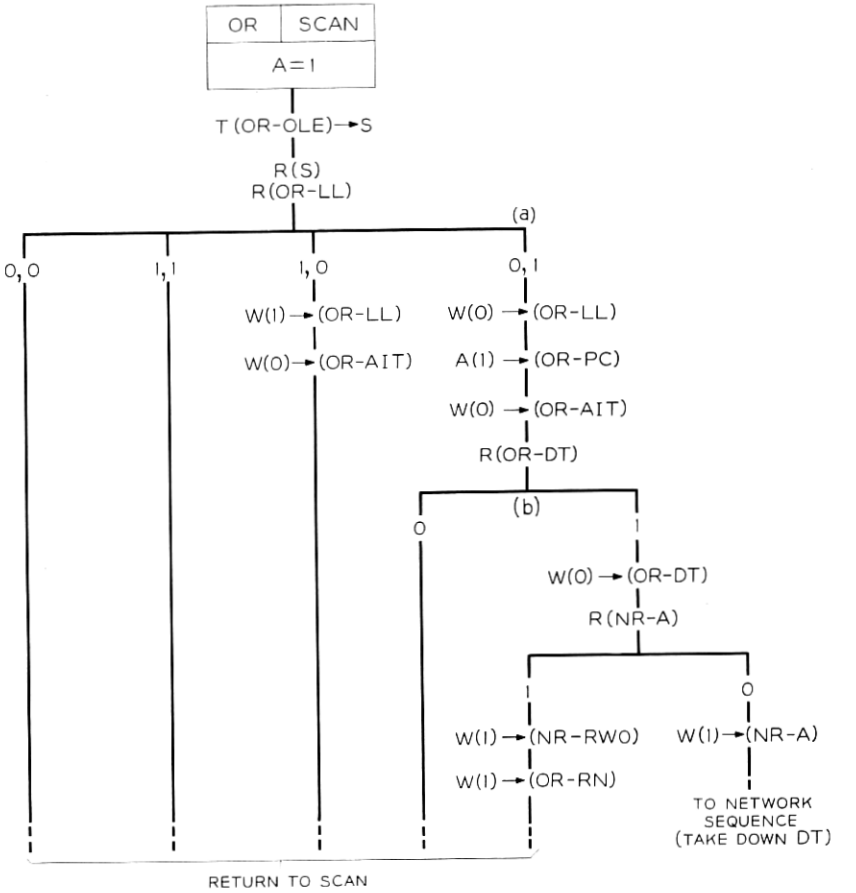


Fig. 7 — Originating register dial pulse scan.

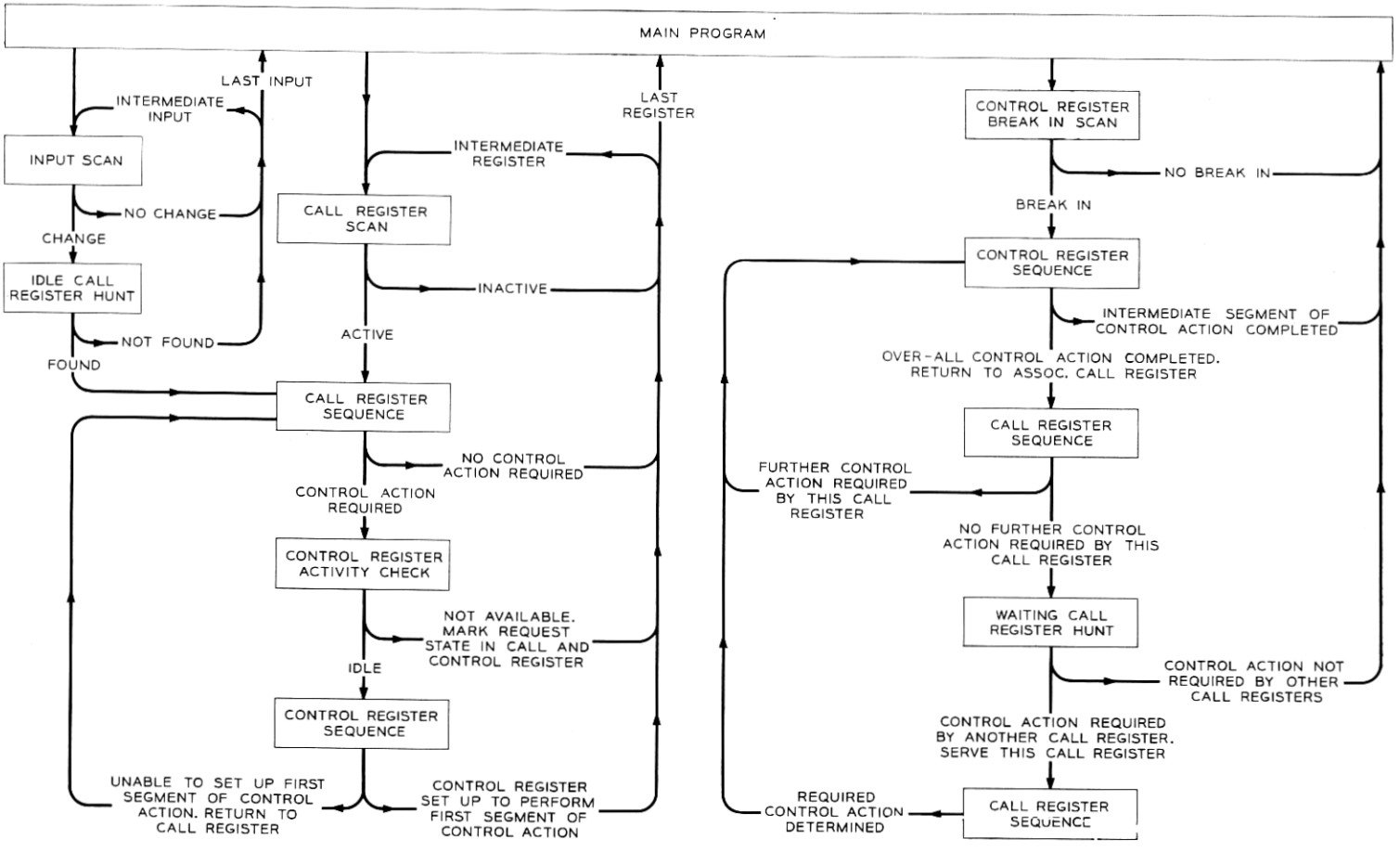


Fig. 8 — General interrelationship of system sequences.

previously discussed for making a network connection. The remaining leg of Fig. 7, when the responses from scanner and BGS are (1,0) respectively, is followed after the trailing edge of a dial pulse is detected.

This sequence (Fig. 7) only accomplishes the detection and storage of dial pulses. Another one recognizes the end of each digit, stores the accumulated pulse count in the appropriate digit storage slot and prepares the originating register for the next digit. Still other sequences carry out the various tasks required to complete the call.

The previous examples are typical of the multiplicity of call sequences or flow charts that are necessary to describe and specify the system operational requirements. They provide a convenient tool which permits the designer to optimize the use of equipment and real time, keep an intelligible record of what the system does and also put the requirements into a form that can be readily programmed.

V. GENERAL INTERRELATIONSHIP OF SYSTEM SEQUENCES

From the few system sequences illustrated thus far, it is evident that the experimental system is not equipped with specific control circuits or programs that are individually associated with a telephone call as it progresses through the system. Instead, the necessary control functions are provided by a single common equipment unit on a time-shared basis as directed by a hierarchy of programs performing a multiplicity of fundamental operations. There is, however, a general mode of system operation which clearly emerges out of the interrelationships among the system sequences.

There are several classes of jobs specified by the system sequences. The more important ones are listed in Table IV with a brief explanation. The relationships among these tasks form the basic operational pattern of the experimental switching system. Fig. 8 illustrates this in block form.

A detailed example of the system action taken, if the left leg of Fig. 8 is followed, can be obtained by tracing through the sequences shown in Figs. 3, 4 and 5. The main program periodically directs the system to determine if any calls have originated or terminated in a group of customer lines, using an "input scan" (top of Fig. 3). If a particular line shows no change, the sequence proceeds to interrogate the next line, and so on. When the last line in the group has been served, the sequence reenters the main program to determine the next task required. Should a change be found on a line, an "idle call-register hunt" (Fig. 4) is made. If no idle register is found the sequence proceeds to the next line or the main program. If, as in the usual case, an idle call register is found, a "call register sequence" is entered (up to point *a* on Fig. 5). At this point,

TABLE IV — FUNCTIONAL CLASSIFICATION OF SYSTEM SEQUENCES

Sequence	Description	Example
Input Scan	A search or scan of system inputs for new work	Top of Fig. 3 — $R(S)$ $R(L1)$
Call Register Scan	A search or scan of call registers (e.g. originating register) for work	Box at top of Fig. 7
Call Register Sequence	A specialized task associated with a call register	Fig. 7
Control Register Break-In Scan	A periodic examination of an equipment for control action	Fig. 6
Control Register Sequence	A task associated with a control register (e.g. network register) and its corresponding equipment unit	Bottom half of Fig. 5
Main Program	The sequence which provides priorities and timing in assigning all system tasks	—
Idle Call Register Hunt	A search for an idle call register	Fig. 4

control action, such as network orders, may be required. The system then proceeds to check the control register for availability and, if appropriate, it advances to a control register sequence (starting at point *b*, Fig. 5). On some sequences a decision will determine whether control action is required (see point *b* on Fig. 7); on others, it is required unconditionally (point *a*, Fig. 5). The description of the remainder of Fig. 8 is left to inspection. This method of system operation was an outgrowth of efforts to optimize the solution of a specialized problem, but many of the techniques employed can be, and some have been, employed in other real-time machines.

The material presented thus far has illustrated what the system is to do, and how it should be done in terms of the external characteristics of the system equipment units. The specific means for implementing the actions prescribed by the system sequence charts, that is, the control philosophy, is the subject of Part Two of this article.

PART TWO

VI. INTRODUCTION

In developing a control philosophy to govern the sequences of system actions, at least two major alternatives must be considered. The first is the use of electronic logic circuits to cause the required actions to be carried out in proper sequence, similar to the way in which relay circuits control present switching systems. This would involve, to a large extent, the individual tailoring of these logic circuits to requirements specified by

the sequence charts described in Part One. The second alternative is realization through a stored program, whereby the pattern of sequences would be incorporated into the program, with the control circuitry designed to interpret and carry out the different types of basic program orders.

In the digital computer field stored programs are used where flexibility is of paramount importance, because of the wide range of applications general-purpose computers must have. Since a telephone switching system is by nature a special-purpose rather than a general-purpose system, the choice between the circuit logic and stored-program approaches was not immediately obvious. The choice of the latter was initially based largely on expected economies of control circuitry and removal of the dependence of the circuit logic on the required sequences of actions. It became increasingly clear, as time progressed, that a still more important aspect of stored-program operation is the flexibility it affords. Of immediate significance in this respect is the ease of addition and revision of system functions; of long-range significance is the real possibility of a universal control for telephone switching systems which could adapt them for local, tandem or toll use as dictated by the stored program.

A stored program for an electronic telephone switching system is a set of encoded orders specifying exactly what the system must do at all times under all possible customer input situations. It exercises exclusive control over the entire system. The detailed functional design of a stored-program electronic switching system, after the control philosophy and required sequences have been established, may be divided into three phases: (1) enumerating the types of basic operations to be incorporated as program orders and determining an encoding structure for these, (2) designing the logic circuitry for their interpretation and execution and (3) writing programs containing these orders to fulfill the requirements of system operation as set forth on the sequence charts. These phases are developed in the following sections.

VII. DEVELOPING A PROGRAM ORDER STRUCTURE

7.1 *Enumeration of Required Operations*

From examination of the general pattern of system operation and the sequence charts already described, it is evident that the system must be equipped to perform certain specific operations. These may be first classified as *decision* and *nondecision* operations.

A decision operation occurs at a branch point of a sequence chart and

provides for a binary choice of alternative actions to follow it. In this category may be included:

1. Read BGS at specified address.
2. Read specified flip-flop.
3. Read scanner at specified address.
4. Match two binary words (determine whether they are equal).

A decision operation is further specified by the addition of a modifier which determines which of two alternatives is to correspond to each of the two possible results. One of the alternative actions following the decision point will be to continue with the next operation appearing sequentially in the program. The other will be to transfer control to some other (predetermined) location in the program.

A nondecision operation performs work called for as a result of some previous decision operation. A nondecision operation is always followed directly by the succeeding operation specified in the program. Included in this category may be:

5. Write 1 or 0 in BGS at specified address.
6. Write 1 or 0 (set or reset) specified flip-flop.
7. Read BGS at specified address (nondecision) and write result in specified flip-flop.
8. Read specified flip-flop (nondecision) and write result in BGS at specified address.
9. Store a specified constant (usually a binary word representing an address) in a specified group of flip-flops.
10. Gate the information in one specified flip-flop group to a second specified flip-flop group.
11. Transfer control from the present program location to a specified program location.
12. Add 1 to the binary number stored in a flip-flop group.
13. Regenerate the BGS at a specified address.

7.2 *Encoding the Program Orders*

From the 13 basic types of operations listed above may be formed an encoding structure for the specific program orders yet to be derived. The process of encoding consists of determining the number of bits a program order of fixed size will contain and what the function of each bit will be.

It is apparent from the operation types that a program order generally contains an instruction (*what*) and an address (*where*). A first step in deriving a coding structure is to classify the addresses required by each operation type, as follows:

1. Read BGS. The BGS contains a square array of 128-by-128 stor-

age locations. Access to one of these therefore requires an address of 14 bits. This erasable bulk memory is organized so that call registers and other aggregates of temporarily stored information are arranged generally within either columns or rows of the array. In this system the BGS will thus be addressed most frequently on successive usages to locations within a single column or row. Therefore, it is logical to precede such a series of BGS operations by presetting a constant row or column address in flip-flop group memory. This will require program orders dealing with the BGS to contain only a seven-bit column or row address rather than a full 14-bit address. Most BGS reading orders will thus contain a seven-bit address, the other seven bits being stored in flip-flop memory.

2. Read Flip-Flop. The size of address required in this type of operation depends only on the anticipated number of flip-flops which must be individually read. Since this number will be in the range between 64 and 128, a seven-bit address is required.

3. Read Scanner. It is required that an order to read the scanner output be preceded by storing the desired address in a flip-flop register associated with the scanner. The Read Scanner operation itself contains no address.

4. Match. This operation determines whether two binary words stored in flip-flop groups are equal. A simple approach to this is to require that one of the two words be stored in a reference flip-flop group with which a matching circuit is associated. The other word may be stored in any one of a number of other flip-flop groups. The matching operation then must include only the address of the latter flip-flop group. Because it was determined that between 16 and 32 flip-flop groups are required for various parallel-access, short-term storage functions, a Match operation requires a five-bit address.

5. Write 1 or 0 in BGS. Same as Read BGS (seven-bit address).

6. Write 1 or 0 in Flip-Flop. Same as Read Flip-Flop (seven-bit address).

7. Read from BGS to Flip-Flop. This operation requires two addresses, a seven-bit partial BGS address (the other seven bits preset in a flip-flop group) and a flip-flop address. For circuit economy, however, 32 of the maximum of 128 individually addressable flip-flops (item 2 above) will suffice to store directly information taken from the BGS. Thus a five-bit flip-flop address accompanies the seven-bit BGS address.

8. Read from Flip-Flop to BGS. Same as above (seven-bit BGS address and five-bit flip-flop address).

9. Store Constant in Flip-Flop Group. It is convenient in this case to

consider the constant as the address-part of this operation. For a number of reasons, involving mostly the detailed numbering plan of the entire system, most of the flip-flop groups contain 14 flip-flops. This operation, if accomplished by a single program order, would therefore contain a 14-bit "address". Since the operation will be restricted to only a small number of flip-flop groups for circuit economy, it is convenient to consider the identity of the flip-flop group in which the constant is to be stored as part of the instruction rather than as an address.

10. Gate Flip-Flop Group to Flip-Flop Group. This operation contains two five-bit flip-flop group addresses.

11. Transfer. An operation causing an unconditional transfer to some point in the program may (a) contain the full flying spot store address to which the transfer is to be made, or (b) specify the identity of a flip-flop group which contains the transfer address. In the latter case, a five-bit address is required; in the former, a 14-bit address is required, assuming the program to be limited to a portion of the flying spot store plates so that 14 bits will afford complete access.

12. Add 1. If we assume that, for circuit economy, a parallel Add 1 circuit is associated with one particular flip-flop group rather than with all flip-flop groups, then the Add 1 operation will cause the binary word contained in that flip-flop group to be incremented by one. Thus, no address is contained in this operation.

13. Regenerate BGS. Since the BGS will be periodically regenerated by rows or columns, this operation need contain only a seven-bit address, the other seven bits being previously stored in a flip-flop group.

TABLE V

Operation Type		Address Requirements	
No.	Name	Number of Addresses	Size (Bits)
1	Read BGS	1	7
2	Read Flip-Flop	1	7
3	Read Scanner	0	
4	Match	1	5
5	Write in BGS	1	7
6	Write in Flip-Flop	1	7
7	Read BGS to Flip-Flop	2	7, 5
8	Read Flip-Flop to BGS	2	5, 7
9	Store Constant	1	14
10	Gate	2	5, 5
11(a)	Transfer	1	14
11(b)	Transfer	1	5
12	Add 1	0	
13	Regenerate BGS	1	7

The address requirements just developed are summarized in Table V. With this information it is possible to establish tentatively the number of bits required by program orders to be derived from the 13 operation types. There are two operation types requiring 14-bit addresses, Transfer and Store Constant. A single program order will be derived from the Transfer operation which contains a 14-bit flying spot store address. In the case of the Store Constant operation, however, it will be expedient to permit a 14-bit constant to be stored in any one of *three* different flip-flop groups; three program orders will therefore be derived from this operation type.

There are, then, four program orders requiring a 14-bit address; thus, two additional bits must be added to distinguish among them. If 16 bits are consumed by only these four program orders, 17 is certainly a lower bound on the number of bits required in a fixed-size program word, since the remaining program orders must also be encoded. As will be seen, 17 bits is sufficient as well as necessary and is adopted as the program word size.

Proceeding with the assumption of a 17-bit word size, the information in Table V is converted to the form of Table VIA to begin the encoding process. Here the addresses included in each operation type are assigned bit positions, starting from the right-hand side. For those cases where two addresses exist (7, 8, 10), the two are placed adjacent, except for operation type 10, where the two five-bit addresses are placed as shown so that the left-hand one is lined up with those of 7 and 8 to achieve as unified a layout as possible.

TABLE VIA

Operation Type		Program Word Bit Position																
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	Read BGS											≡	≡	≡	≡	≡	≡	≡
2	Read Flip-Flop											≡	≡	≡	≡	≡	≡	≡
3	Read Scanner													≡	≡	≡	≡	≡
4	Match											≡	≡	≡	≡	≡	≡	≡
5	Write in BGS											≡	≡	≡	≡	≡	≡	≡
6	Write in Flip-Flop											≡	≡	≡	≡	≡	≡	≡
7	Read BGS to Flip-Flop						≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡
8	Read Flip-Flop to BGS						≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡
9	Store Constant				≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡
10	Gate						≡	≡	≡	≡	≡			≡	≡	≡	≡	≡
11(a)	Transfer				≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡
11(b)	Transfer													≡	≡	≡	≡	≡
12	Add 1																	
13	Regenerate BGS											≡	≡	≡	≡	≡	≡	≡

TABLE VIb

	Operation Type	Program Word Bit Position																
		A			B		C					D						
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	Read BGS											≡	≡	≡	≡	≡	≡	≡
2	Read Flip-Flop											≡	≡	≡	≡	≡	≡	≡
3	Read Scanner													≡	≡	≡	≡	≡
4	Match											≡	≡	≡	≡	≡	≡	≡
5	Write in BGS											≡	≡	≡	≡	≡	≡	≡
6	Write in Flip-Flop											≡	≡	≡	≡	≡	≡	≡
7	Read BGS to Flip-Flop						≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡
8	Read Flip-Flop to BGS						≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡
9	Store Constant				≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡
10	Gate						≡	≡	≡	≡	≡			≡	≡	≡	≡	≡
11(a)	Transfer				≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡
11(b)	Transfer													≡	≡	≡	≡	≡
12	Add 1																	
13	Regenerate BGS											≡	≡	≡	≡	≡	≡	≡

Table VIb repeats the information in VIa, with the lines of demarcation bounding the addresses darkened so that the 17-bit word is subdivided into groups of three bits (called A), two bits (B), five bits (C) and seven bits (D). In places where a five-bit address occupies the rightmost positions [4, 10, 11(b)], no subdivision is shown because it will prove convenient circuit-wise to treat these as seven-bit addresses with the two most significant bits zero.

A program order word may now be thought of as containing an A code, a B code, a C code and a D code. For orders derived from operation 1 above, for example, the BGS seven-bit partial address will be contained in the D code, while the A, B and C codes together will specify the exact nature of the operation to be performed. For operation 11(a), on the other hand, the B, C and D codes together will contain the flying spot store address to which a transfer is to be made, while the A code will direct that a transfer be made.

A process of encoding has now been described except for the detailed specification of all the program orders to be derived from the 13 operation types and the illustration of how these fit into the proposed structure. This final phase will be deferred to a later section which deals with the writing of programs. In that section a complete set of program orders will be presented.

At this point we turn to a discussion of the circuit logic of the central control, whose function is to interpret and carry out all the types of 17-bit program orders provided.

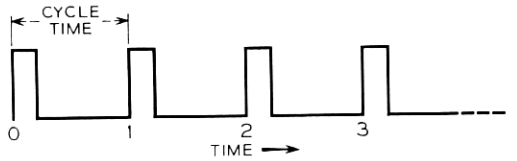
VIII. THE CENTRAL CONTROL

8.1 *Basic Considerations*

The central control is the heart of the electronic switching system. Its actions are required for all system functions to be accomplished. It consists almost entirely of direct-coupled semiconductor circuitry[7] arranged in logic configurations to accomplish the interpretation and execution of each program order originating from the flying spot store (FSS).

Central control processes these orders one at a time. The time allotted to carry out an order is dependent upon the speeds of the major system components, in particular the flying spot store, barrier grid store and central control itself. The progress through the program, order by order, is controlled by a clock whose period is constant and sufficiently long to cover the operating time of any of the above-mentioned units. This time will be referred to as the "cycle time" and is of several microseconds duration.

When a program order is passed from the FSS to central control on the occurrence of a clock pulse, the central control logic circuits, within the ensuing cycle time, decode the order and prime those parts of the central control which are to participate in the execution of the order. The succeeding clock pulse causes the execution of the order and, at the same time, causes the next order in the program to pass from the FSS to the central control for similar processing, as shown in Fig. 9. This "overlap" arrangement between the FSS and central control is used to conserve system time. The order-by-order progression just described is altered in the case of decision orders, as will be discussed later.



TIME	ACTION
0	ORDER A ENTERS CENTRAL CONTROL
1	ORDER A IS EXECUTED; ORDER B ENTERS CENTRAL CONTROL
2	ORDER B IS EXECUTED; ORDER C ENTERS CENTRAL CONTROL
3	ORDER C IS EXECUTED; ORDER D ENTERS CENTRAL CONTROL
ETC.	ETC.

Fig. 9 — Electronic switching system clock timing.

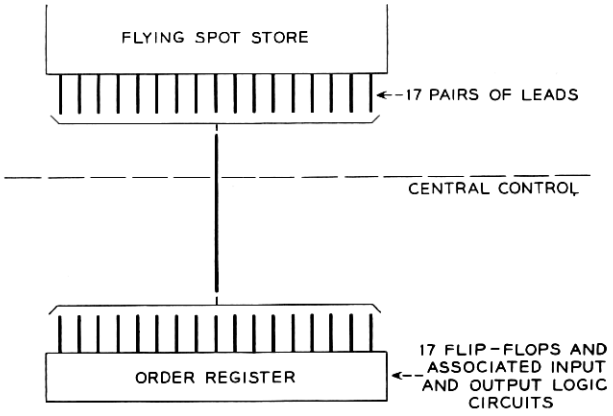


Fig. 10 — Central control order register.

Since the FSS output is detected by means of a short sampling pulse, the first requirement in central control is for a register to receive and then hold the order for a cycle time. This register is called the order register and is shown in Fig. 10. It consists of 17 flip-flops, one to hold each bit of the order, and logic circuits to properly associate the register with the circuits connected to it. The first component of the cycle time consumed during the processing of an order is the interval from the beginning of a clock pulse to the time at which the outputs of the flip-flops and amplifiers of the order register correctly indicate the bits of the order; this is approximately one-third of the cycle time.

Since the order is binary-encoded, a decoding or translating process is next required, to convert the order into a form which can be used for establishing the circuit conditions required for its execution. The association of an order translator used for this purpose with the order register is shown in Fig. 11. The input to the translator is a 17-bit order; its output usually consists of signals on a few of more than 100 leads. The translator logic circuits and amplifiers themselves require about one-third of a cycle time to respond, so that, by the time the proper translator outputs are active, approximately two-thirds of the cycle time has expired. The remaining one-third of the cycle time is consumed by those logic circuits that act in response to order translator outputs. Clock pulses control circuit actions only at the input to central control (between the FSS and the order register) and at the output stage of logic circuits where the execution of orders is accomplished.

The order translator outputs are connected in some manner to almost

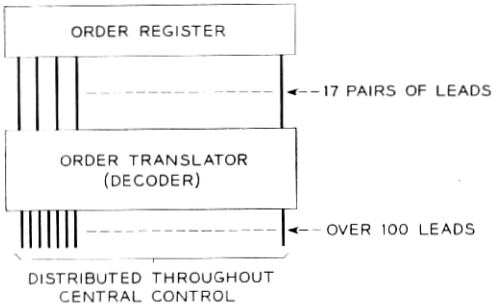


Fig. 11 — Central control order translater.

every circuit of the central control.* Much of this circuitry consists of flip-flop registers, including special-purpose registers associated with specific functions and general-purpose registers with multifunctional application. The registers are employed mainly for short-term high-speed storage of addresses associated with the various major system units. The communication among these flip-flop groups, and between them and circuits external to central control, may be achieved through the use of a common bus, since the program orders permit only one parallel aggregate of bits to be communicated in a single cycle time. In addition to establishing conditions for communications via the bus, the order translator outputs also control actions internal to a register circuit or actions associated with circuits having control rather than register functions.

Fig. 12 shows a simplified picture of the relationships among the order register, order translator, flip-flop group registers, bus, some control circuits yet to be described and circuits external to central control. All of the registers attached to the bus need only contain 14 or fewer flip-flops if it is assumed that this number is sufficient to specify an address in any major system unit. As shown, some registers have communications both to and from the bus; all of the general-purpose and some special purpose registers are in this category. Other special-purpose registers have communications either to the bus or from the bus but not both, most of these being directly associated with external system units.

The bus itself consists of 14 pairs of OR gates whose outputs serve as the common communication path. For a 14-bit register having communication to and from the bus, the paired outputs of its 14 flip-flops can be gated to the inputs of the 14 flip-flops of any other register so connected.

* It is beyond the scope of this article to cover completely the many detailed functions of central control. It is intended, rather, to develop and explain only the more fundamental features.

To clarify this arrangement and to illustrate how a simple program order is carried out by central control, an example is given which involves only those circuits already discussed. Let us examine how the order "Gate the information in flip-flop group A to flip-flop group B" (operation type 10, Table V) is accomplished. The logic circuit actions resulting from this order are shown in Fig. 13. At time 0 in this example a program order enters central control from the FSS coincident with a clock pulse and is held by the order register. As time elapses, the flip-flops of the order register respond to the order and the order translator logic responds to the outputs of these flip-flops. For the order "Gate FFG A (flip-flop group A) to FFG B", two translator outputs are activated, one causing the outputs of the flip-flops of FFG A to be gated to the bus input, the other readying the bus output to be gated to the inputs of the flip-flops of FFG B. Thus, within a cycle time following time 0—that is, before the occurrence of the next clock pulse at time 1—the information held by FFG A will have been transmitted via the bus to gates at the input of FFG B. Then, at time 1, coincident with the next clock

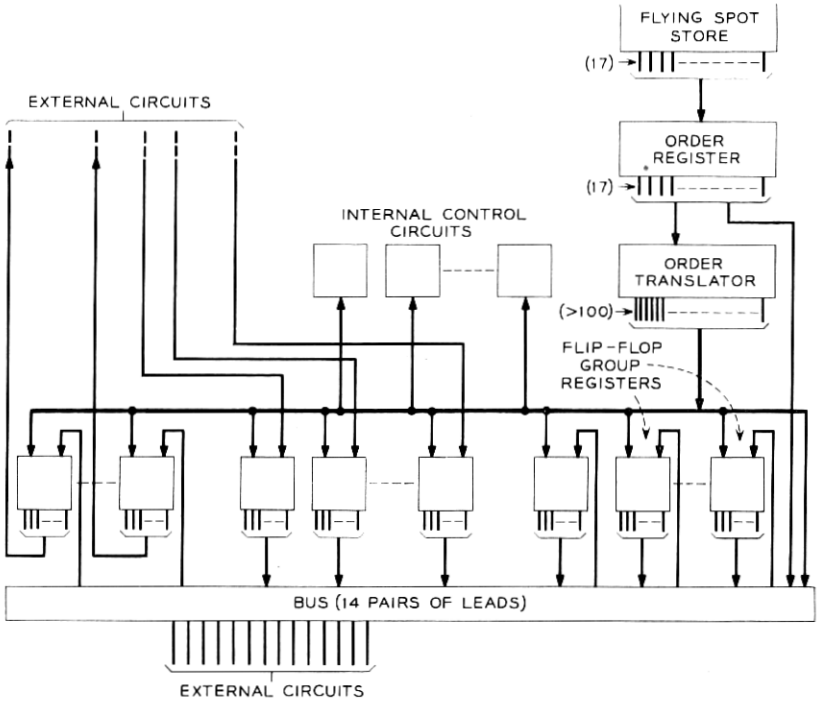


Fig. 12 — Simplified block diagram of central control.

pulse, this information is gated into FFG B and the next order enters the order register from the FSS.

A second example, which brings into play an external system unit, is shown in Fig. 14. Here the order to be processed, derived from operation type 5, is "Write 1 in the BGS at an address whose Y part (vertical coordinate of the 128-by-128 BGS array) is the D code of the order and whose X part (horizontal) is stored in flip-flop group Q". Here the order translator activates four of its outputs, which (a) gate the D code order register flip-flop outputs to the Y half of the bus input, (b) gate the X half flip-flop outputs of FFG Q to the X half of the bus input, (c) ready the bus output to be gated to the BGS address leads and (d) ready

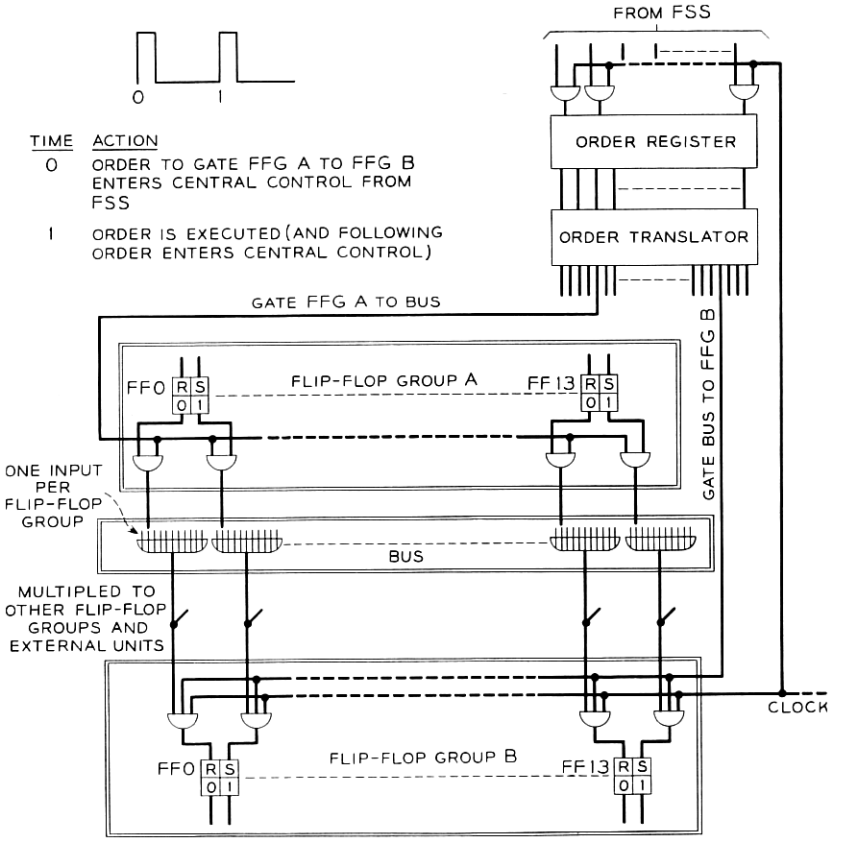


Fig. 13 — Central control circuits for simple nondecision order: "Gate FFG A to FFG B."

a Write 1 signal to be gated to the BGS. When the clock pulse occurs at time 1, this order is executed.

8.2 *Decision Making*

Both of the examples just presented involved nondecision operations; that is, the Gate FFG to FFG and Write 1 in BGS orders can be followed only by the succeeding order in the program, located at the next FSS address. We will now develop the decision-making logic which comes into play on decision (reading or matching) operations when the following

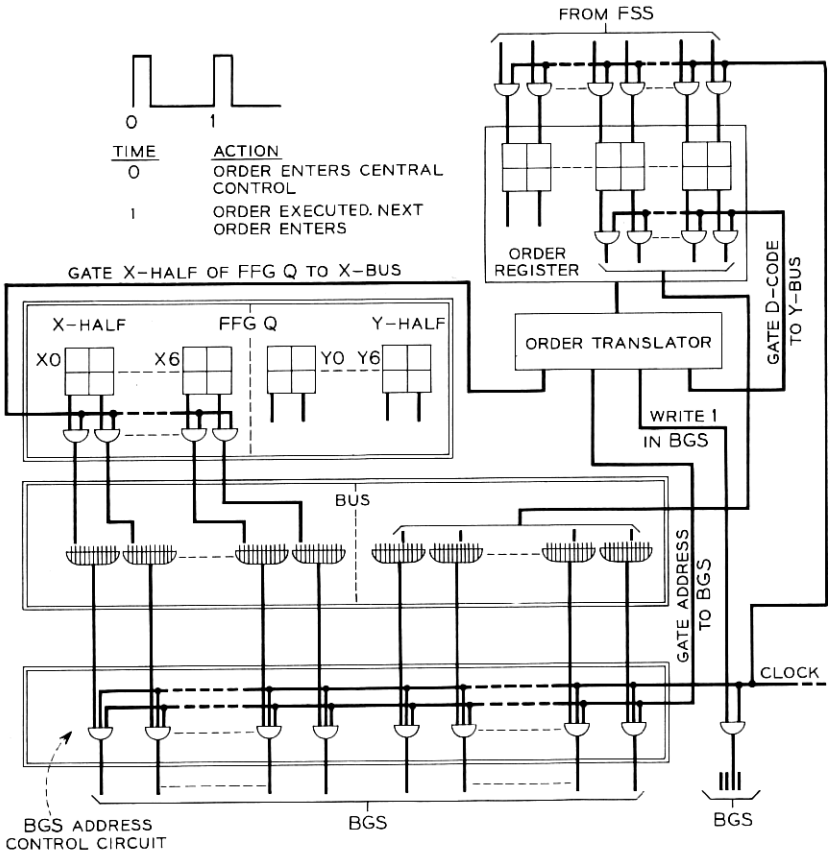
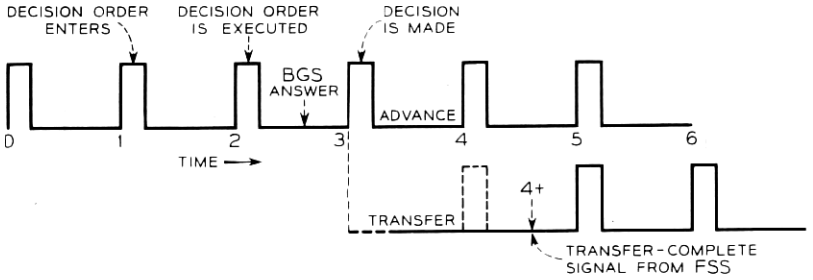


Fig. 14 — Central control circuits for another nondecision order: "Write 1 in BGS at Y address given in D code of order, X address preset in X half of flip-flop group Q."

action is one of two alternatives, and discuss the way the clocking of the system is affected.

A complication arises where decisions are involved because, at the same time an order is executed, it is displaced from the order register by the succeeding order. Since the response to a decision operation occurs *after* the execution of that operation, means must be provided for recovering certain basic information in the order which will determine, when combined with the response, which of two alternative actions is to follow. This point is illustrated in Fig. 15 which shows the timing involved when a Read BGS order (described in detail below the figure) is performed; it should be noted that, if the BGS response is 0, the system continues with



TIME	ACTION
0	NON-DECISION ORDER A ENTERS CENTRAL CONTROL, FSS GOES TO ORDER B
1	ORDER A EXECUTED; (DECISION) ORDER B ENTERS CENTRAL CONTROL; FSS GOES TO ORDER C
2	ORDER B EXECUTED; ORDER C ENTERS CENTRAL CONTROL; FSS GOES TO ORDER D

IF READING IS 0:

TIME	ACTION
3	ORDER C EXECUTED; D ENTERS; FSS TO E
4	ORDER D EXECUTED; E ENTERS; FSS TO F
5	ORDER E EXECUTED; F ENTERS; FSS TO G
ETC	ETC

IF READING IS 1:

TIME	ACTION
3	ORDER C NOT EXECUTED; FSS DIRECTED TO TRANSFER TO ADDRESS STORED IN FLIP-FLOP GROUP T
4+	FSS SIGNALS CENTRAL CONTROL THAT TRANSFER IS COMPLETE
5	ORDER J ENTERS CENTRAL CONTROL; FSS GOES TO ORDER K
6	ORDER J EXECUTED; ORDER K ENTERS CENTRAL CONTROL; FSS GOES TO ORDER L
ETC	ETC

Fig. 15 — Central control timing on decision orders. The decision order, designated order B, is: "Read (and regenerate) the BGS at the address whose Y part is D the code of the order and whose X part is preset in flip-flop group Q. If the reading is 0, continue with the next order (c) in the program; if the reading is 1, transfer to order J, which is another point in the program whose FSS address is (was previously) stored in flip-flop group T."

the next order in the program, whereas if it is 1, a transfer is made. At time 2 the Read BGS order (designated order B) is executed; that is, the Read-Regenerate lead and an address are pulsed to the BGS; also at Time 2, order B is displaced from the order register by order C. The BGS response is returned to central control during the interval between time 2 and time 3. Clearly, if a decision is to be made at time 3 whether or not to transfer, the specification contained in the order as to which sense of BGS response should cause a transfer must be *stored* when the order is displaced at time 2. Then, at time 3, this partial memory of the order may be combined with the BGS response to determine the decision. Here, if the BGS response is 0, order C, which follows order B in the program and which entered the order register at time 2, is executed. If the BGS response is 1, order C is not executed but, instead, a transfer signal is sent together with an address to the FSS. If we assume hypothetically that the FSS requires between one and two cycle times to reach the transfer address and read out the order J stored there, a transfer-complete signal[5] will be returned to central control at time 4+. The effect of the clock pulse occurring at time 4 is inhibited because at that time the transfer has not yet been completed. However, the transfer-complete signal at time 4+ permits the clock pulse at time 5 to bring order J from the FSS into the order register and causes the FSS to go on to order K. At time 6, J is executed, K enters the order register, and the FSS goes on to L, and so forth.

The addition of the order memory and decision logic used in conjunction with decision orders is shown for the previous example in Fig. 16. Here five order translator outputs are activated: one gates the X half of FFG Q to the X bus; a second gates the D code of the order register to the Y bus; a third readies the bus output to be gated to the BGS as an address; a fourth prepares a Read-Regenerate order to be gated to the BGS and a fifth prepares the "Transfer if 1" flip-flop in the order memory to be set. The latter three actions will occur at time 2 (Fig. 15), when the clock pulse passes through the decision logic and exits on the "Execute Present Order" (EPO) lead through gates not shown. Also at time 2, order B will be displaced from the order register by order C, but not before the order memory flip-flop is set to retain the sense of the transfer specified in order B. The BGS response occurs between time 2 and time 3. At time 3, therefore, if the BGS response is 0 an EPO pulse will be generated, causing order C to be executed. (The nature of order C is not specified and the circuits shown should not be assumed to be adequate to process it). If the BGS response is 1, on the other hand, a "Conditional Transfer" (CTR) pulse is generated by the decision logic,

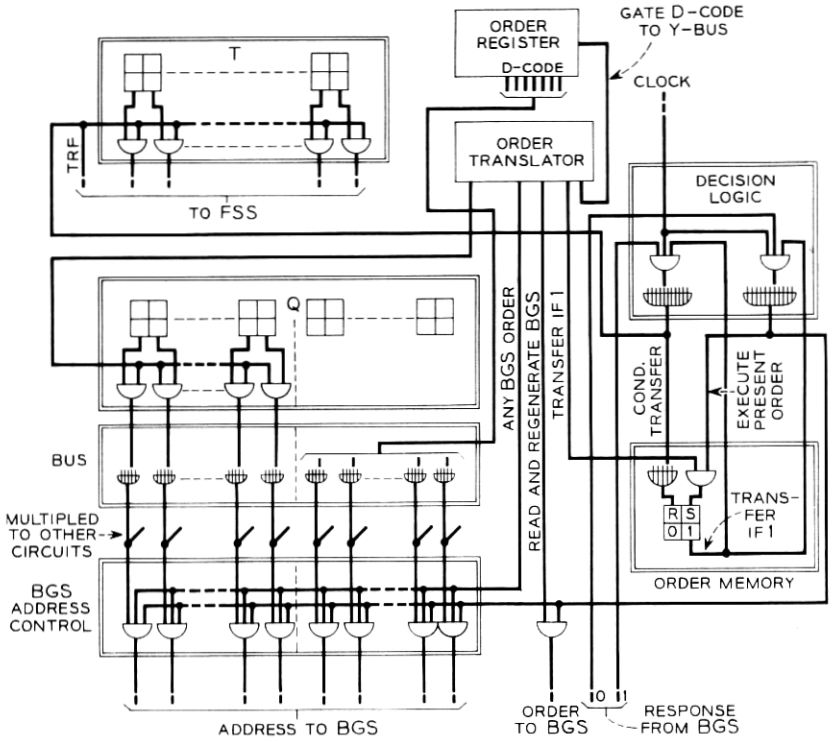


Fig. 16 – Central control circuits for simple decision order: "RY1."

which resets the order memory and causes a transfer signal to be sent to the FSS, together with the address previously stored in FFG T. In general, every clock pulse is converted either to an EPO pulse, which causes whatever order is in the order register to be executed and displaced by the succeeding one, or a CTR pulse, which causes a transfer of the FSS to a new order located at a previously stored address.

8.3 *Functional Description of Central Control**

With the basic features of central control design now described, we may consider the more complete functional diagram shown in Fig. 17.

* The material in this section describes in some detail the interrelationships among the several registers and control circuits of the central control. If the reader wishes, he may proceed at this point directly to Section IX (Programming) without loss of continuity.

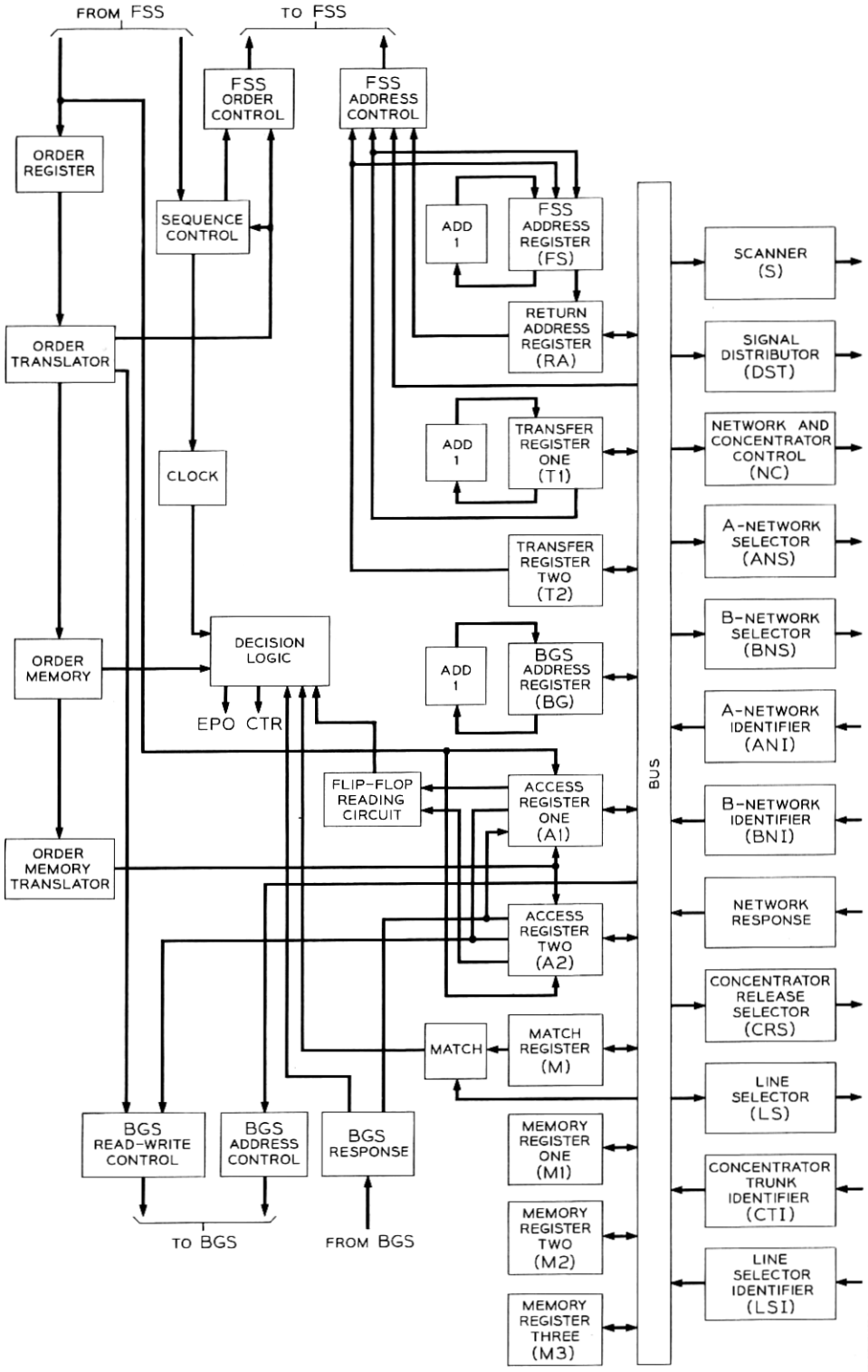


Fig. 17 — Functional block diagram of central control.

8.3.1 *Special-Purpose Registers*

On both sides of the bus in Fig. 17 are shown flip-flop groups which will hereafter be referred to as "registers." On the right of the bus are special-purpose registers which have access either to or from the bus, but not both. Those registers which are used only to communicate control or address information to external system units from central control receive information from the bus output; those used to pass information from external units to central control are connected to the bus input.

In the first class are the following:

1. Scanner Register. Whenever a Read Scanner order is given, this register must already hold the address of the line or trunk whose state is desired.

2. Signal Distributor Register. This holds the address of the relay to be operated or released by the signal distributor.

3. Network and Concentrator Control Register. The code stored in this register instructs the concentrator and distribution network marker circuits what actions to carry out. The addresses involved are stored in other registers.

4. A-Side Network Selector Register. This holds the A-side distribution network address to be used in establishing or releasing a distribution network connection.

5. B-Side Network Selector Register. Same as above for B-side.

6. Concentrator Release Selector Register. This holds the concentrator trunk address when a concentrator connection is released.

7. Line Selector Register. This holds the line address when a line-to-concentrator trunk connection is to be made through the concentrator.

All these registers may be considered functional parts of the external units to which they are connected, since the outputs of their flip-flops directly control the actions in these units. They are made physically a part of central control to avoid the need for transmitting pulses to them from the bus over the distances between central control and external units. With this arrangement, only dc flip-flop outputs must travel over long lead lengths.

The remaining registers shown to the right of the bus in Fig. 17 have access to the bus input, passing information from the network to central control. These are:

8. A-Side Distribution Network Identifier Register. Whenever a connection including an A-side terminal is either established or released this register will store the address of this terminal.

9. B-Side Distribution Network Identifier Register. Same as above for B-side.

10. Concentrator Trunk Identifier Register. This receives a partial address of the concentrator trunk involved in a concentrator connection or release operation.

11. Line Selector Identifier Register. This receives the line address on a concentrator connect operation, or a partial address of a concentrator trunk on a concentrator release operation. The combination of the information in this and the previous register specifies the full address of a concentrator trunk on a release operation, and of a line and concentrator trunk on a connect operation.

12. Network Response Register. The flip-flops in this register receive indications of whether a network action is ended and whether it is successful or unsuccessful.

8.3.2 *General-Purpose Registers*

To the left of the bus in Fig. 17 are registers which have access both to and from the bus. Some are special-purpose but are available for general-purpose use when their special functions are not required. Others are provided only for general-purpose use. Their functions are described below:

13. Flying Spot Store Address Register. This is included here to facilitate descriptions of other registers, although it has access neither to or from the bus. At all times this register holds the FSS address of the program order awaiting execution in the order register. An Add 1 circuit is associated with the FSS address register so that, as each order is executed and followed by the next order in the program, the address it holds may be incremented. When a transfer occurs, the FSS address to which the transfer is to be made is sent to this register as well as to the FSS.

14. Return Address Register. Frequently a program is used repeatedly and, because of this, is recorded in the FSS as a subroutine. When a program transfers into a subroutine it is often necessary that the program be resumed following the completion of the subroutine, i.e., that the subroutine end with a transfer returning to the program which initiated it. The return address register receives from the flying spot store address register, upon each transfer, the FSS address following the one from which the transfer is made. Thus, a subsequent transfer to the FSS address stored in this register will cause the program from which the original transfer was made to be resumed.

15. Transfer Registers 1 and 2. These hold the FSS address to which a transfer may be made as the result of a decision order. The decision order must specify from which of the transfer registers the address is taken. The reason that two are provided and the function of the Add 1 circuit associated with one of them will be explained in a later section.

16. Barrier Grid Store Address Register. This supplies part or all of a BGS address on all BGS reading and writing orders. As already explained, BGS orders provide at most a seven-bit X or Y address, the remaining seven bits being taken from a flip-flop register. This was the register referred to and which also appeared in the examples of Figs. 14 and 16 as flip-flop group Q. The Add 1 circuit that is associated with it is provided for the Add 1 operation listed in Table V. If a number is to be incremented by 1, it is first placed in the BGS address register and then acted upon by the Add 1 circuit. Thus the register has two special-purpose uses and may be employed for general-purpose use as well.

17. Access Registers 1 and 2. These are the most versatile registers of the central control. They are characterized by the fact that their flip-flops are individually accessible. As indicated by the many paths to and from these registers in Fig. 17, they are used: (a) for communications to and from the bus; (b) when a bit is read from the BGS to be stored in a flip-flop (operation type 7, Table V); (c) when a bit is read from a flip-flop to be stored in the BGS (operation type 8); (d) when a bit is to be read from or written into an individual flip-flop (operation types 2 and 6); (e) to receive translation information from the flying spot store, since individual bits of a translation word must be examined.

18. Matching Register. This is used for the matching operation (type 4). In it is placed a number to be matched with another number stored in any other register with access to the bus. The associated Match circuit compares the two numbers and notifies the decision logic of the result.

19. Memory Registers 1, 2 and 3. These are completely general-purpose registers used most frequently to store program addresses for later use.

The registers having intercommunications by way of the bus may be thought of as a rectangular matrix of flip-flops, one dimension being the number of such registers and the other the maximum number of flip-flops per register (14). Not all elements of the matrix are occupied, especially in the case of some special-purpose registers associated with external units where the number of bits required is fewer than 14. Clearly, the flip-flops in such registers must be assigned definite posi-

tions on the bus so that proper communications can be carried on with the remainder of central control.

8.3.3 *Control Circuits*

On the left side of Fig. 17 are shown the circuit blocks which have control rather than register functions. Among them are the order register, order translator, order memory, decision logic and clock, all of which have already been covered in earlier paragraphs.

Associated with the order memory is an *order memory translator*, whose outputs affect the two access registers. This translator is required for operation type 7, which involves a BGS reading being stored in a flip-flop of one of the access registers. As shown in Table VIb, the five-bit C code identifies this flip-flop. Since the BGS response occurs after the order has been displaced from the order register, the C code must be held in the order memory and translated by the order memory translator to a one-out-of-32 indication to prepare the input logic of the appropriate access register flip-flop for receipt of the BGS response.

The three circuits associated directly with the BGS and shown near the bottom of the figure are the *read-write control*, *address control* and *response circuit*. The BGS read-write control determines which, if any, of the four BGS orders is to be given (Read and Regenerate, Read and Write 0, Read and Change, Read and Write 1); most often the order translator alone determines this, but, in the case of operation type 8, the state of an access register flip-flop will determine whether a Write 0 or Write 1 signal is given. The BGS address control gates the bus output to the BGS address leads as shown on Figs. 14 and 16. The BGS response circuit output is connected to the decision logic for determining whether or not to transfer on BGS decision operations, and to the access registers for storing a bit read from the BGS into a flip-flop for later use.

The *flip-flop reading circuit* shown in Fig. 17 is essentially a large OR gate which combines the outputs of all the flip-flops of central control which can be individually read (using operation type 2). During the processing of a Read Flip-Flop order, the seven-bit flip-flop address is converted by the order translator to a signal which permits only the output of the corresponding flip-flop to be transmitted into the flip-flop reading OR gate. In this way the state of this flip-flop appears at the output of the OR gate.

The *flying spot store address control* at the top of Fig. 17 receives a FSS address from one of a number of possible sources and transmits it to the FSS when a transfer is made. Included among these sources are

the two transfer registers, the return address register and the bus. The FSS address of a transfer also is stored in the FSS address register of central control, as already explained.

The *flying spot store order control* generates the orders to the FSS. Except for a few special situations, whenever the decision logic converts a clock pulse to an Execute Present Order pulse, an Advance pulse is sent to the FSS, which causes the succeeding order to be gated from an output register in the FSS to the order register in central control, and then causes this same output register to be filled with the next order. When the decision logic converts a clock pulse to a Conditional Transfer pulse, this circuit sends a Transfer signal to the FSS and the FSS accepts a new address from central control rather than stepping along, as it does in the case of an Advance.

The last block yet to be described in Fig. 17 is the *sequence control.* This is the only basically sequential circuit in the central control. It functions whenever control of the system is temporarily taken away from the program or whenever completely synchronous clock control gives way to asynchronous control actions. The two major uses of this circuit are in the cases of transfers and FSS translations.

The time required to complete a transfer is somewhat variable over a small range because of the characteristics of the FSS.[5] The sequence control therefore acts to inhibit clock pulses following the one on which a transfer occurs from affecting the central control until it receives a Transfer-Complete signal from the FSS. It causes the clock pulse following this signal to gate the order at the transfer location from the FSS to the order register and then returns normal synchronous control to the clock.

In the case of a FSS translation, a transfer is made to an address where a translation word, not an order, is to be read out and stored directly in the access registers. Clearly, a separate control circuit is necessary to accomplish this, since the FSS cannot be used simultaneously to control the system with an order and to act as a source of translation information. Here, the sequence control steers the FSS output into the access registers rather than the order register. It then causes a second transfer to the address held by the return address register, causing control to be returned to the program order following the one which initiated the FSS translation.

This completes a functional description of central control. All functions are realized through the use of diode AND and OR gates, flip-flops and inverters. Added to this basic logic are amplifiers, emitter-followers and other supporting circuits. A central control like the one

described would contain on the order of 2000 transistors and 15,000 diodes.

IX. PROGRAMMING

This final section deals with the writing of the programs which will be recorded on the photograph plates of the flying spot store. A program is a group of orders, each encoded into a 17-bit word, which specifies in complete detail what the system is to do. From the programmer's viewpoint, the nature of the central control and other circuit action occurring in the execution of the order is of little concern. It is essential that the programmer know only the action taken as a result of each order, and, in all cases, this is specified by its definition.

This section of the article will, therefore, have little direct relation to the previous one. Its purpose is to explain how electronic switching system programs are composed and, in the process, to illustrate a few of the problems confronting the programmer.

9.1 *A Detailed Order Structure*

Before programs can be written there must exist a complete list of available program orders, each precisely defined. The basis for such a list was established earlier and is shown in Table VIb. It is now appropriate to derive from this list of operation types a set of program orders. Considerations related to those discussed earlier are used in arriving at variations of the operation types which will prove to be useful orders. The patterns of system actions depicted on the sequence charts are especially helpful in forming judgments as to the potential utility of a proposed order. Ease of circuit implementation is also a contributing factor.

After a tentative list of orders is composed and some programs are written, orders which are found to have infrequent application and can be replaced by combinations of other orders are deleted; others may be added as a result of discoveries of useful applications by programmers. A stable order structure eventually emerges.

The order structure to be used in this article is shown in Table VII. The listing includes 38 orders, 16 decision and 22 nondecision. For each order is shown (a) a symbolic designation having mnemonic significance, (b) the operation type in Table VIb from which it is derived, (c) its numerical (ABCD) coding in decimal form and (d) an abbreviated but precise definition.

TABLE VII

No.	Symbol	Operation Type (from Table VIb)	A, B	C	D	Description
			Decision Orders			
1	RY-	1	*	0	—	Read and regenerate BGS at Y specified by D (X given by BGX).
2	RX-	1	*	1	—	Read and regenerate BGS at X specified by D (Y given by BGY).
3	EY-	1	*	2	—	Read and erase BGS at Y specified by D (X given by BGX).
4	EX-	1	*	3	—	Read and erase BGS at X specified by D (Y given by BGY).
5	CY-	1	*	4	—	Read and change BGS at Y specified by D (X given by BGX). Suffix refers to reading before change.
6	CX-	1	*	5	—	Read and change BGS at X specified by D (Y given by BGY). Suffix refers to reading before change.
7	RP-	1†	*	6	—	Read and regenerate the BGS at address stored in BG.
8	RYB-	1	*	7	—	Read and regenerate at Y specified by D (X given by BGX). Store Y in BGY.
9	RXB-	1	*	8	—	Read and regenerate at X specified by D (Y given by BGY). Store X in BGX.
10	EYB-	1	*	9	—	Read and erase at Y specified by D (X given by BGX). Store Y in BGY.
11	EXB-	1	*	10	—	Read and erase at X specified by D (Y given by BGY). Store X in BGX.
12	RFF-	2	*	11	—	Read miscellaneous flip-flop specified by D.
13	RS-	3	*	12	—	Read the scanner at address preset in S.
14	MY-	4†	*	13	—	Match the Y bits of M with the Y bits of the FFG specified by D. Match = 1; mismatch = 0.
15	MX-	4†	*	14	—	Match the X bits of M with the X bits of the FFG specified by D. Match = 1; mismatch = 0.
16	MB-	4	*	15	—	Match the contents of M with the contents of the FFG specified by D. Match = 1; mismatch = 0.
			Decision Order Modifier Suffixes			
—	-0	—	0, 0	—	—	If 0 is read, take next order from address given by T1.
—	-1	—	1, 0	—	—	If 1 is read, take next order from address given by T1.
—	-02	—	0, 2	—	—	If 0 is read, take next order from address given by T2.
—	-12	—	1, 2	—	—	If 1 is read, take next order from address given by T2.
—	-0A	—	0, 1	—	—	Add 1 to Y part of contents of T1. Then, if 0 is read, take next order from address given by T1.
—	-1A	—	1, 1	—	—	Add 1 to Y part of contents of T1. Then, if 1 is read, take next order from address given by T1.

TABLE VII — *Continued*

No.	Symbol	Operation Type (from Table VIb)	A, B	C	D	Description
			Nondecision Orders			
17	W0Y	5	7, 3	0	—	Write a 0 on the BGS at the Y address specified by D (X given by BGX).
18	W1Y	5	7, 3	1	—	Write a 1 on the BGS at the Y address specified by D (X given by BGX).
19	W0X	5	7, 3	2	—	Write a 0 on the BGS at the X address specified by D (Y given by BGY).
20	W1X	5	7, 3	3	—	Write a 1 on the BGS at the X address specified by D (Y given by BGY).
21	W0P	5†	7, 3	4	—	Write a 0 on the BGS at the address contained in BG.
22	W1P	5†	7, 3	5	—	Write a 1 on the BGS at the address contained in BG.
23	W0FF	6	7, 3	6	—	Write a 0 in the miscellaneous flip-flop specified by D.
24	W1FF	6	7, 3	7	—	Write a 1 in the miscellaneous flip-flop specified by D.
25	RYFA	7	3, 0	—	—	Read and regenerate the BGS at the Y address specified by D, and transport the bit to the access flip-flop specified by C (X given by BGX).
26	WFAY	8	3, 1	—	—	Transport the contents of the access flip-flop specified by C to the BGS at the Y address specified by D (X given by BGX).
27	ST1	9	4, -	—	—	Set up transfer register 1 to the number specified by B, C, D.
28	SA1	9	5, -	—	—	Set up access register 1 to the number specified by B, C, D.
29	SA2	9	6, -	—	—	Set up access register 2 to the number specified by B, C, D.
30	SY	9†	7, 3	10	—	Set up BGY to the number specified by D.
31	SX	9†	7, 3	11	—	Set up BGX to the number specified by D.
32	G	10	3, 3	—	—	Gate the contents of the FFG specified by C to the FFG specified by D.
33	T	11a	2, -	—	—	Transfer to the address specified by B, C, D.
34	TFG	11b	7, 3	12	—	Transfer to the address contained in the FFG specified by D.
35	AY	12	7, 3	13	—	Add 1 to BGY.
36	AX	12	7, 3	14	—	Add 1 to BGX.
37	RGY	13	7, 3	15	—	Regenerate the BGS at the Y address specified by D (X given by BGX).
38	RGX	13	7, 3	16	—	Regenerate the BGS at the X address specified by D (Y given by BGY).

* Modifier code.
† Size of address not same as specified in Table VI.

A decision order must be suffixed with a modifier indicating which sense of response should cause a transfer. There are six modifiers: (a) "0" suffixed to a decision order (e.g., RY0) means transfer on a 0 response to the FSS address stored in transfer register 1; (b) the "1" suffix causes the transfer on a 1 response; (c) and (d) the 02 and 12 suffixes are the same as those above, respectively, except that the transfer address is taken from transfer register 2; (e) and (f) the 0A and 1A modifiers are similar to 0 and 1 except that the FSS Y address held by transfer register 1 is incremented by 1, the purpose of this to be illustrated by a later example. Thus, by adding one of the six modifiers to a decision order it is possible to cause a transfer on either a 0 or 1 response to the FSS address stored in either transfer register 1 or 2, and, if transfer register 1, either changing or not changing the Y part of the address stored there.

The 16 decision orders are derived from the first four operation types. The RY-* order causes the BGS to be read and regenerated at the Y address specified by the D code and the X address preset in the X half of the barrier grid register of central control. The RX- does the same, except that the D code is the X address and Y is preset. The EY- and EX- orders are similar, but the BGS is read and erased instead of regenerated. The CY- and CX- cause the BGS to be read and changed; here, for decision purposes, the response is assumed to be the reading before the change occurs. The RYB-, RXB-, EYB- and EXB- are the same as RY-, RX-, EY- and EX- except that the D code is gated to the Y or X half of the barrier grid register in central control as well as to the BGS as a Y or X address. This is provided for those cases where it is useful to keep track in the barrier grid register (BG) of the BGS address last visited. The RP- order causes the BGS to be read and regenerated with the entire address preset in BG, the order itself containing no address.

The MY- order causes the contents of the Y half of the register specified by the D code to be matched against the contents of the Y half of the match register. The 1 suffix is used for the "match" response and the 0 suffix for "mismatch". MX- accomplishes the same for the X halves and MB- for both halves together, or the entire contents of both registers.

The RFF- and RS- orders are the only ones derived from operation types 2 and 3 and their definitions are self-explanatory.

The 22 nondecision orders are derived from operation types 5 through

* The hyphen stands for a modifier, which must be attached.

13 of Table VI, and their descriptions in Table VII require no amplification.

9.2 *The Register Matrix*

Combinations of the program orders described above are used to effect all system actions, many of which involve the central control together with some other system unit. Because the major system components (Fig. 1) differ in function and size, a "numbering plan" is formed which ties them together in proper association. In central control, the numbering plan is reflected to some extent in the relationship of each flip-flop register to the bus. Table VIII shows the bus having 14 positions divided into X and Y coordinates. The general-purpose registers occupy the full 14 positions, whereas the registers associated with other system units are tailored, both with respect to size and placement on the bus, to the numbering plan.

If a 12-bit scanner address were to be taken from the BGS and placed in the scanner register via access register 1, for example, the 12 bits

TABLE VIII

Register (Abbreviations shown in Fig. 17)	X Position							Y Position						
	Bus													
	X_6	X_5	X_4	X_3	X_2	X_1	X_0	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
A1	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A2	27	26	25	24	23	22	21	20	19	18	17	16	15	14
BG	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡
BGX*	≡	≡	≡	≡	≡	≡	≡							
BGY*								≡	≡	≡	≡	≡	≡	≡
T1	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡
T2	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡
RA	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡
M	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡
M1	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡
M2	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡
M3	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡
S		≡	≡	≡	≡	≡	≡		≡	≡	≡	≡	≡	≡
DST			≡	≡	≡	≡	≡			≡	≡	≡	≡	≡
ANS						≡	≡	≡	≡	≡	≡	≡	≡	≡
BNS						≡	≡	≡	≡	≡	≡	≡	≡	≡
CRS					≡	≡	≡	≡	≡	≡	≡	≡	≡	≡
LS		≡	≡	≡	≡	≡	≡		≡	≡	≡	≡	≡	≡
ANI						≡	≡	≡	≡	≡	≡	≡	≡	≡
BNI						≡	≡	≡	≡	≡	≡	≡	≡	≡
CTI					≡	≡	≡							≡
LSI			≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡
NC			≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡	≡
NR									≡	≡	≡	≡	≡	≡

* BGX and BGY together comprise BG but are separately addressable.

would have to be placed from the BGS into access flip-flops 0, 1, 2, 3, 4, 5, 7, 8, 9, 10, 11 and 12. Access register 1 could then be gated to the scanner register, since these flip-flops occupy the same bus positions as the 12 flip-flops of the scanner register.

9.3 *Examples of the Uses of Program Orders*

In this section several examples of the uses of program orders are given. Each example contains a sequence to be programmed, any BGS layout involved in the sequence and one or more programs fulfilling the requirements of the sequence. In the examples the symbolic languages for sequences and programs already presented will be employed. Frequent reference to Table VII will be helpful.

Example 1 — Fig. 18

The sequence in this example requires the writing in two BGS locations having different Y addresses (A, T) and a common X address (R1). The corresponding program contains three orders. The order SX = R1 is used to place BGS X-address R1 in the BG register of central control (see Table VIII); this is necessary since orders to read or write in the BGS contain at most an X *or* Y address, the other half being preset in BGY or BGX, respectively. The orders W1Y = A and W0Y = T are used to write 1 in R1-A and 0 in R1-T, A and T being the Y addresses specified in these two orders with X preset as a result of the previous order. The = sign in a program order separates the operation part of the order from the address(es) at which the operation is carried out.

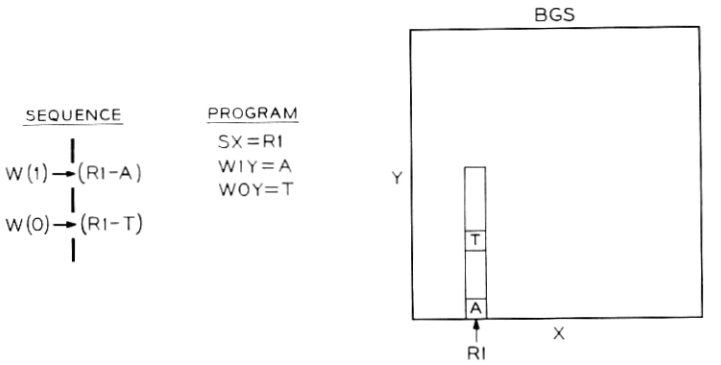


Fig. 18 — Programming example 1.

Example 2 — Fig. 19

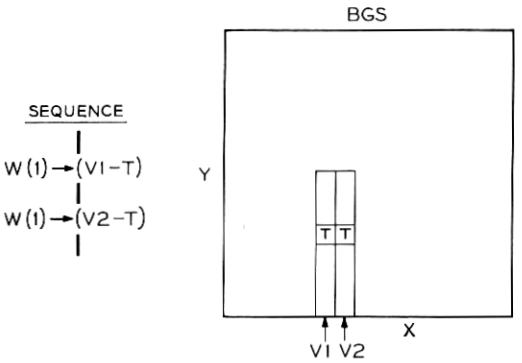
The sequence here requires writing 1 at two BGS locations having a common Y address (T) and different X addresses (V1, V2). Three alternative programs are shown.

Program (a) is analogous to the previous example, except that X and Y are interchanged and 1 is written at both BGS locations.

Program (b) begins with the orders SY = T and SX = V1, placing the entire BGS address of V1-T in the BG register. The order W1P causes a 1 to be written at this address. Then SX = V2 and W1P cause a 1 to be written at V2-T.

Program (c) is similar to (b) except that the BGS address V1-T is placed in the BG register by first placing it in access register 1 (A1) and then gating the contents of A1 to BG. Thus the orders SA1 = V1, T (which places V1 in the X half and T in the Y half of A1) and G = A1, BG accomplish this. The remaining three orders are identical to those of (b).

Program (a) requires the fewest orders (three) and would therefore be used. Programs (b) and (c) are shown to introduce the use of the W1P, SA1 and G orders.



ALTERNATIVE PROGRAMS

(a)	(b)	(c)
SY=T	SY=T	SA1=V1,T
W1X=V1	SX=V1	G=A1,BG
W1X=V2	W1P	W1P
	SX=V2	SX=V2
	W1P	W1P

Fig. 19 — Programming example 2.

Example 3 — Fig. 20

This sequence involves a decision based on the reading of BGS location NR-A. If the reading of NR-A is 1, a 1 must be written in NR-RW; if it is 0, NR-A must be changed to 1.

In program (a) the decision order RY1 = A is preceded by ST1 = r, which places in transfer register 1 the address r of the program to be transferred to if the reading of NR-A is 1. In program (b) the transfer is made instead when the reading of NR-A is 0 and the two writing orders following it are interchanged with respect to program (a). The choice between these two programs in this case would be to make the transfer the lower-probability event, since it consumes more time than advancing to the next program order.

Program (c) is similar to (a) with X and Y interchanged. In this case, the group of orders SY = A, ST1 = r, RX1 = NR have exactly the same over-all effect as the corresponding orders of (a). In (c), however,

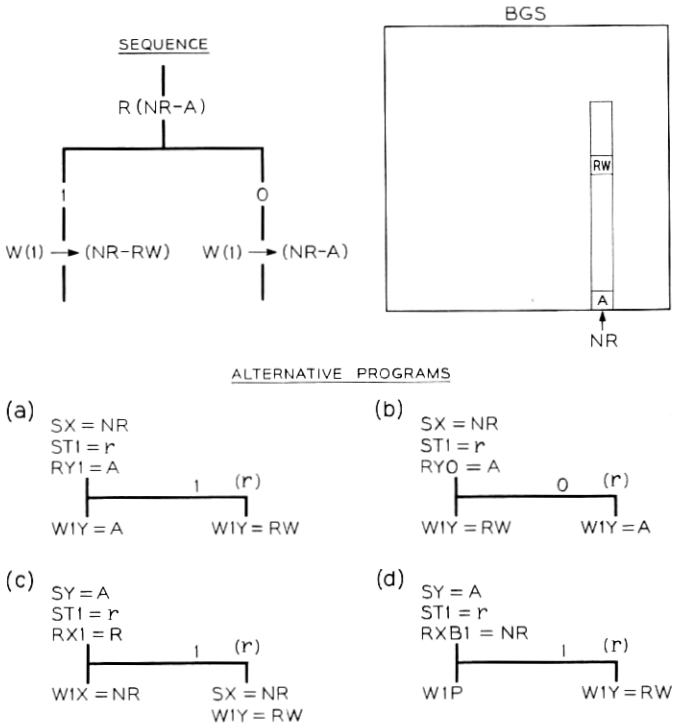


Fig. 20 — Programming example 3.

an additional order (SX = NR) is necessary before a 1 can be written in RW, since the X address NR must be placed in the X half of the BG register before W1Y = RW will have the desired effect.

The additional order is avoided in (d), where the RXB1 order is used in place of RX1. The order RXB1 performs the reading function of RX1, but also causes the address part (NR) of the order to be gated to the X half of the BG register, making unnecessary the order SX = NR used in (c). Also, W1X = NR in (c) is replaced by W1P in (d), since the combination of SY = A and RXB1 = NR result in the full address of NR-A being in the BG register.

Example 4 — Fig. 21

The sequence in this example involves a hunt among a group of BGS registers R1, R2, ··· RN for one in which the A location reads 0. When such is found a 1 is written in the A location and the program reaches

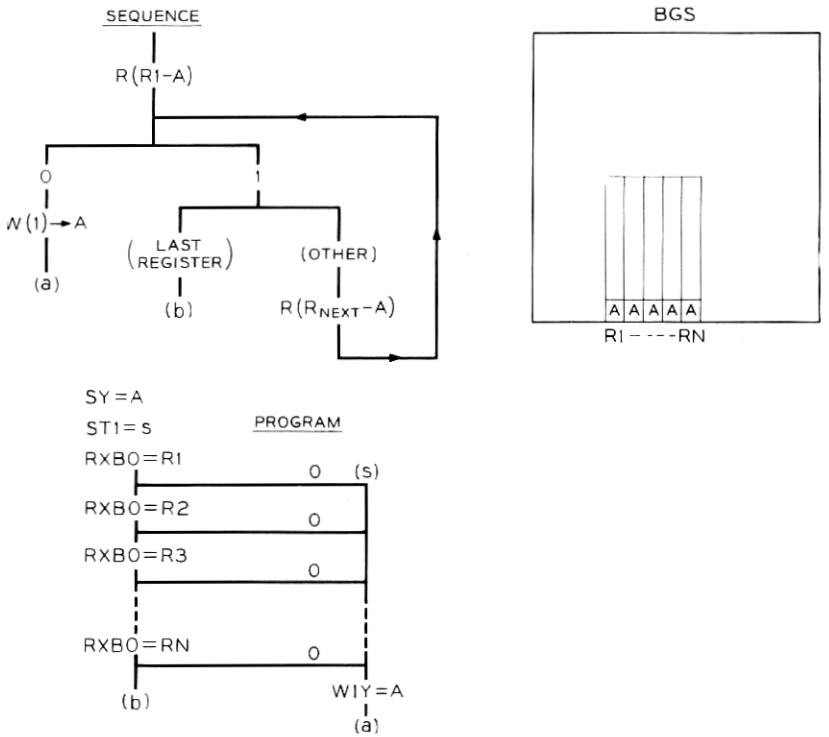


Fig. 21 — Programming example 4.

point *a*. If none is found the program reaches point *b*. Frequent use of the RXB order occurs in this type of program.

Example 5 — Fig. 22

This sequence illustrates the use of the Read-and-Erase (EY and EXB) orders. The sequence requires that a reading be made of NR-A; if the reading is 1, NR-A should be written to 0. Programs (a) and (b) are otherwise similar to programs already described.

Example 6 — Fig. 23

This example illustrates the use of the Read-and-Change (CY) order. The sequence requires that a counter (PC) containing two bits (PCA, PCB) be incremented by 1 followed by a 1 being written into OR1-TR. The incrementing is done by first changing the value of the least significant bit (PCA). If the value of PCA is changed from 0 to 1, the process is finished; if PCA is changed from 1 to 0, the value of PCB is changed and the process is finished.

Program (a) accomplishes this using types of orders already described in previous examples. After setting the X half of the BG register to OR1 and transfer register 1 to *v*, the PCA bit is read (its Y address

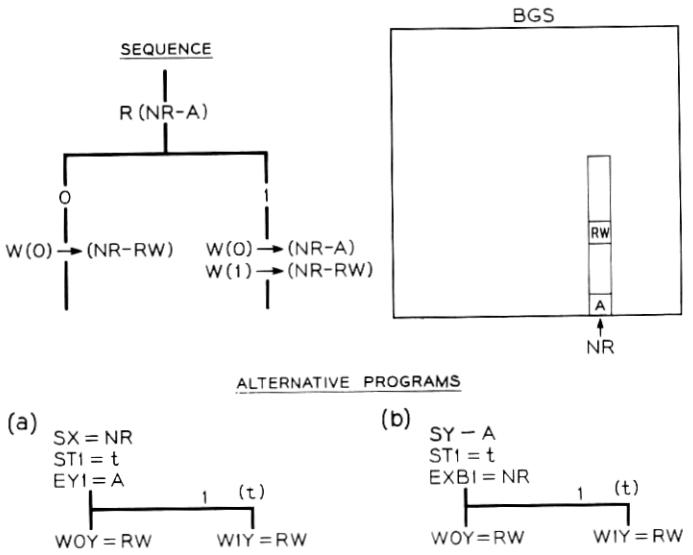


Fig. 22 — Programming example 5.

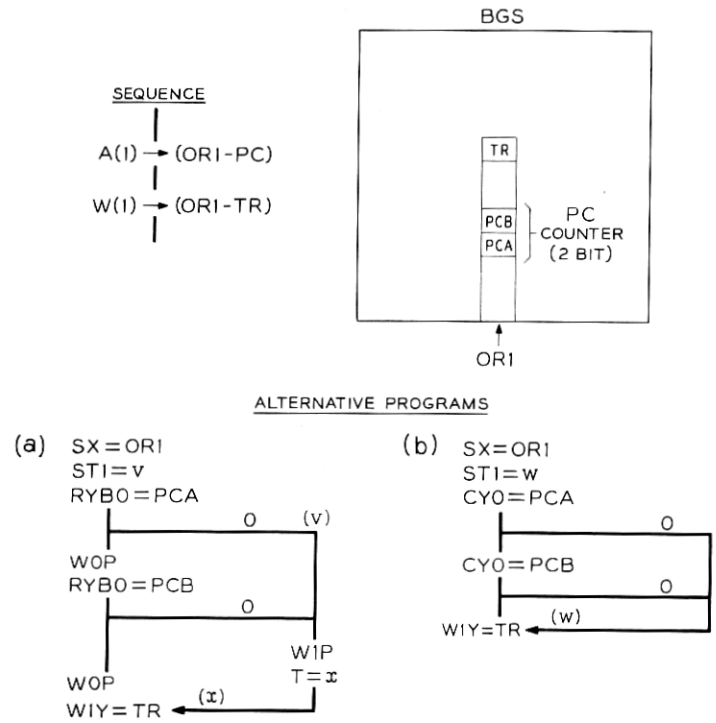


Fig. 23 — Programming example 6.

being gated to the Y half of the BG register since RYB0 is used). If the reading is 0, a transfer is made to v and a 1 is written in PCA by the W1P order; since the process is then finished, a transfer is made by the T = x order to program location x, where the sequence continues with W1Y = TR. If the reading of PCA were 1, on the other hand, it would be written to 0 by the W0P order and PCB would be read with the pattern repeating.

Whereas program (a) contains nine orders, the use of the CY order in program (b) reduces the number of orders to five. The CY0 order both reads a bit and changes its value, combining the functions of RYB0, W0P and W1P in program (a).

Example 7 — Fig. 24

The sequence of this example calls for three successive decisions based on the reading of three BGS locations (NR-P, Q and R). Since

presumably different actions take place on each possible leg of the sequence (a, b, c, d), a different program location must be provided for each.

Program (a) contains seven orders and fulfills the requirements of the sequence in a straightforward manner. Note that a separate ST1 order must be used preceding each RY0 order to provide the different program locations a, b and c.

In program (b) the repeated use of ST1 is eliminated by using the RY0A order (refer to Table VII–decision order modifiers). This order performs the function of RY0 but in addition increments the address in transfer register 1. Thus, program location b becomes related to

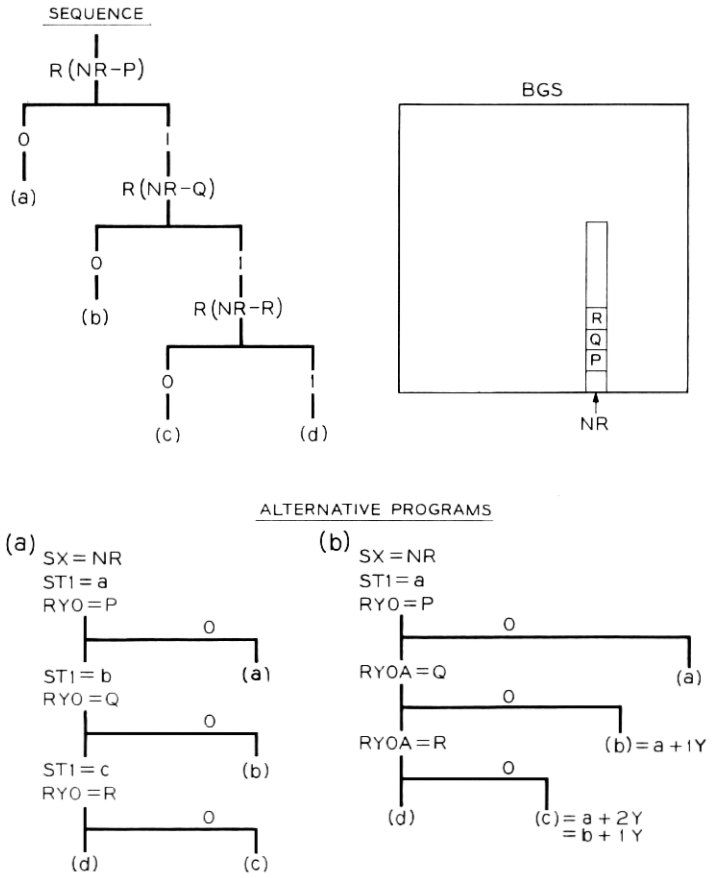


Fig. 24 — Programming example 7.

program location *a* by a difference of 1 in the Y coordinate, and the two may be used as starting points for completely different and unrelated programs. The same applies to program location *c*.

Example 8 — Fig. 25

This example illustrates the programming of a simple subroutine. A subroutine may be thought of as a program required under many and varied circumstances but recorded only once to conserve program space. Thus a subroutine may be transferred into from a number of sources and must, upon completion, cause a transfer back to a place associated with the source involved.

The sequence shown should be considered extracted from a larger sequence and requires that a 1 be written in OR–A1 of the BGS. But there are several OR's, and the X address of the one desired is recorded elsewhere in the BGS, in NR-ORX1 to NR-ORX7. These seven bits must be extracted and transported to the X half of the BG register to steer action to the appropriate one of the several OR's. Since this action occurs repeatedly in different parts of the program, the process of extracting the seven bits from NR is made a subroutine.

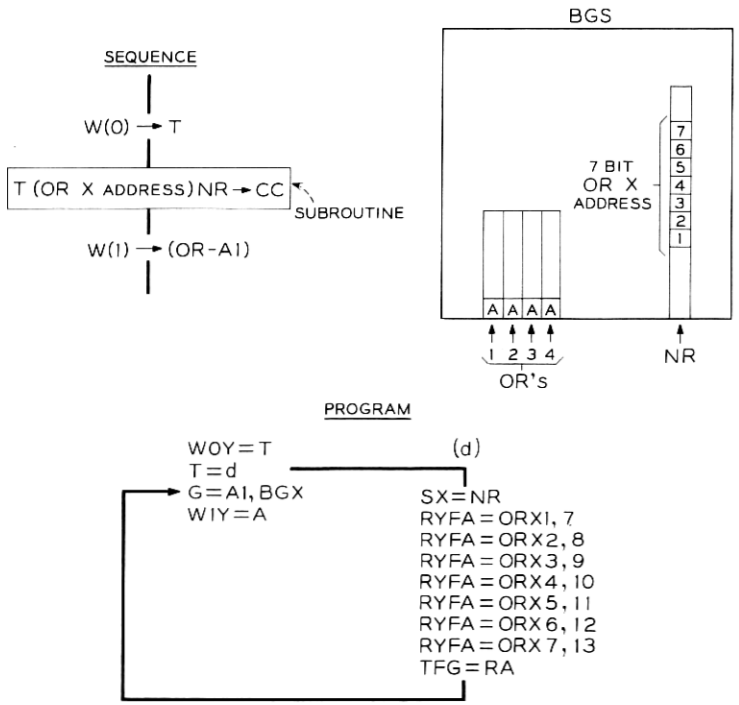


Fig. 25 — Programming example 8.

The program first transfers to the subroutine located at d (T = d). The subroutine begins with SX = NR to confine succeeding BGS operations to the proper X address. This is followed by a series of seven RYFA orders, each reading one of the seven bits from the BGS to a flip-flop in access register 1; the address part of these orders contains both the BGS Y address of the bit being read and the identity of the access register flip-flop into which the bit is placed. It should be noted from Table VIII that the bus positions of the seven flip-flops chosen coincide with those of the X half of the BG register. The subroutine ends with TFG = RA, which causes a transfer back to the return address, one address beyond that from which the transfer to the subroutine was made, as stored in the return address register. The order G = A1, BGX then gates the seven bits from flip-flops of access register 1 to corresponding flip-flops of the BG register, and W1Y = OR-A1 causes a 1 to be written in the A1 location of the proper one of the OR's.

9.4 *Programming a Larger Sequence*

In this section some of the techniques introduced in the foregoing examples are brought together in the programming of the larger sequence shown in Fig. 26*; the associated BGS layout is given in Fig. 27. A program satisfying the requirements of the sequence is given in Fig. 28. Program locations (flying spot store addresses) a, b, c, d, e, f are shown both in the program and on the sequence chart to relate corresponding portions of the two.

Beginning at point a in the sequence, a reading must be made of OR1-A1, a location in the BGS. If it is 0, OR2-A1 is read, and so forth through all the OR's in the BGS. If a 1 is read, on the other hand, the sequence requires 1 to be added to AIT and then a reading to be made of AIT. If it is 1, OR(NEXT)-A1 is read as before; if it is 0 (point b) a reading is made of LL.

The program for this portion of the sequence is shown in Fig. 28 at location a. A pattern of alternation of RXB0 = OR- and CY12 = AIT orders is evident. The RXB0 orders perform the function of reading A1 for each OR; each RXB0 is preceded by ST1, so that if the reading is 0 a transfer is made to repeat the process for the next OR. If the reading on RXB0 is 1, CY12 = AIT is performed, adding 1 to the AIT counter. A transfer to the address in transfer register 2 (point b) will then take place if AIT was 1 and was changed to 0; otherwise, the next OR will be treated. The result of the process to this point is that a transfer to b will occur for each OR satisfying the conditions A1 = 1, AIT = 1 (before the addition of 1, making AIT = 0) as required by the sequence.

* The particular system function performed by the sequence is of no concern in this section and will not be described.

The correspondence between the remainder of the program and the requirements of the sequence is left to the reader.

9.5 *A Typical Programming Problem*

Previous examples have clearly shown that a number of different programs can be written which satisfy the requirements of a given sequence. The programming problem is therefore not primarily one of arriving at a correct program as much as arriving at one which is both correct and optimal. In programming a real-time system like the one described in this paper, two minimization criteria apply: program space (number of orders) and program time (time consumed per unit of real

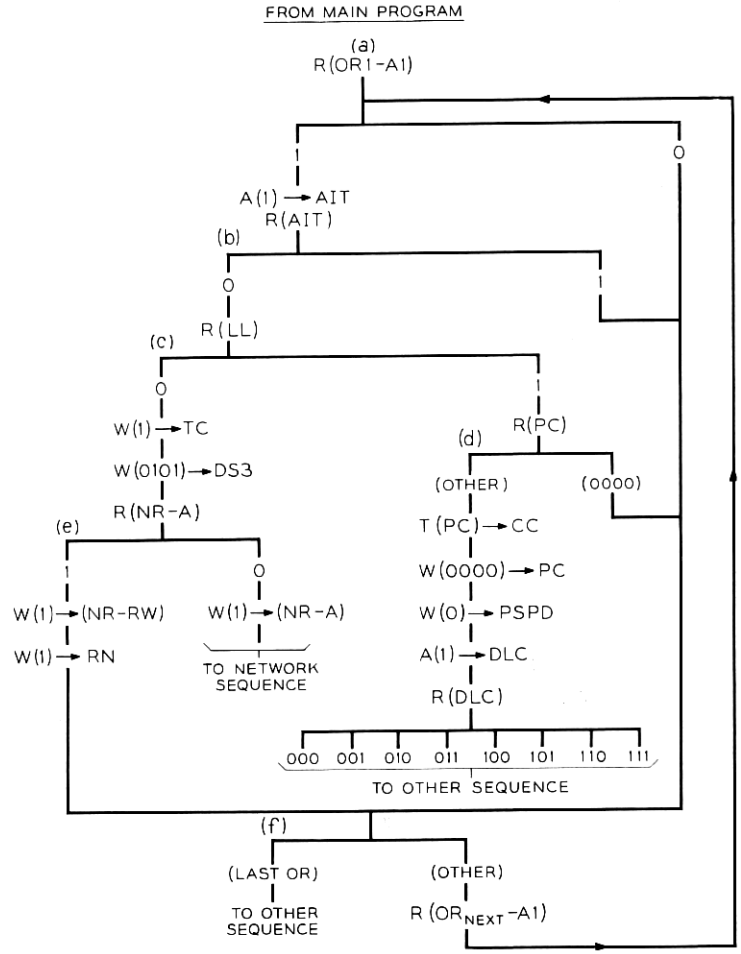


Fig. 26 — A sequence chart to be programmed: Abandoned and interdigital timing (AIT).

time). The optimal program is the one which minimizes some elusive function of these two variables.

In most programming problems the two effects are opposite; i.e., program space reduction is usually accompanied by an increase of program time, and vice versa. A simple example of this is illustrated in Fig. 29. Program (a) contains 65 orders and consumes 65 cycle times; program (b) contains only 11 orders but consumes 259 cycle times.

In general, the programmer combines qualitative and quantitative judgments in deciding the relative merits of alternative courses of action in this and similar situations.

9.6 *The Assembly of Programs*

The writing of programs in symbolic mnemonic form has the valuable advantage of detaching the programmer from unwieldy numerical coding details which would tend to increase the number of errors made. On the other hand, symbolic programs necessitate a process of conversion to the binary form in which programs are actually recorded in the flying spot store.

The latter process, referred to as assembly, is mechanized by the use of a digital computer. Into the computer are placed a symbolic program,

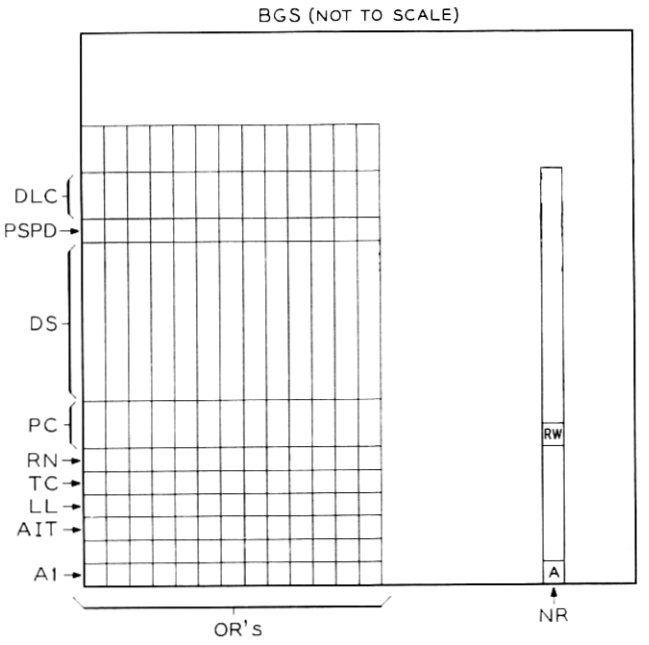


Fig. 27 — Associated BGS layout for AIT scan program.

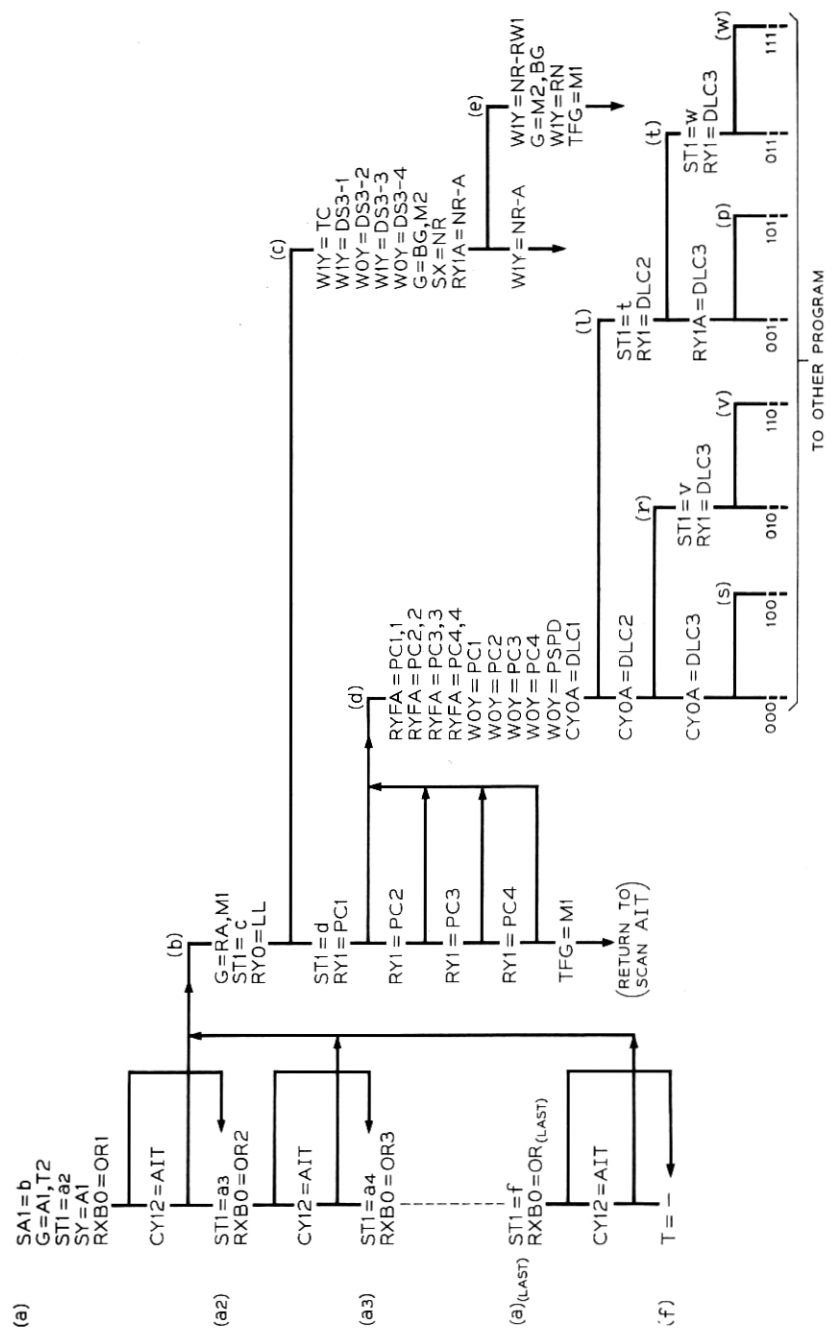


Fig. 28 — Scan program for AIT.

a series of tables specifying the conversions from symbols to binary numbers of the operation-parts and address-parts of program orders, and a set of rules in the form of a computer program for applying the conversions to the symbolic program. Out of the computer comes the binary encoded result ready to control the exposure of the photographic plates of the flying spot store.

9.7 *Simulation*

In the testing of a large stored program system, there is the serious problem of initially distinguishing program errors from equipment troubles. In the case of the experimental electronic switching system an attempt was made to attain an error-free program, so that any difficulties encountered later in laboratory tests of the system could be definitely attributed to equipment.

To accomplish this a large general-purpose digital computer was used to simulate the physical system while the latter was still in its final stage of design. The computer was programmed to image all the memory of the system, accept program orders written in electronic switching system language and interpret and carry out these orders on the system image. In this way, "telephone calls" were processed in the computer and information was printed to indicate the success or failure

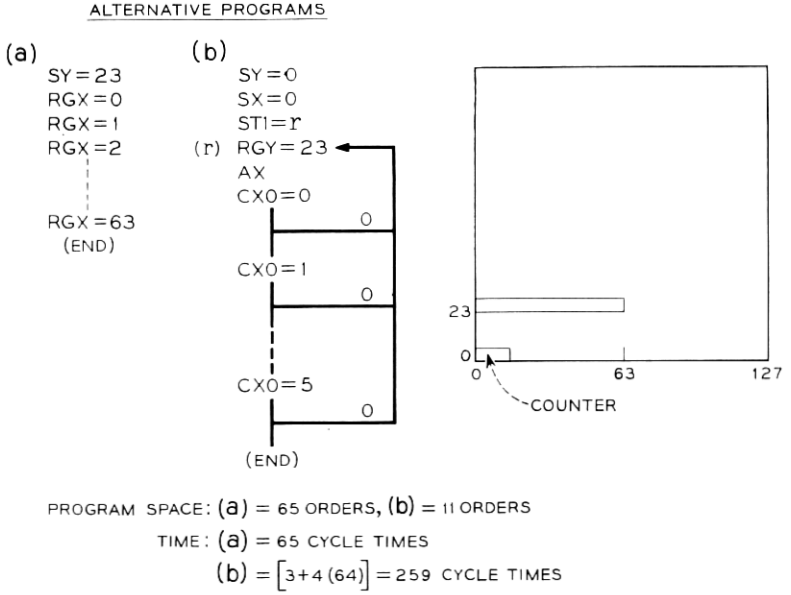


Fig. 29 — Time vs. program space. Sequence: "Regenerate the 64 BGS locations in the left half of row 23. Space for a six-bit counter is available if desired."

of the calls; if the latter, additional information was made available aiding the location of the part(s) of the electronic switching system program causing the failure. After corrections were made based on this information, the process was repeated until the computer reported perfect operation of the program.

X. SUMMARY

This article describes the design of a stored-program electronic switching system from a functional viewpoint. Included are: (1) a description of the major functional units of the system, (2) the introduction of a language and form to describe the sequence of system actions, (3) some examples of how these sequences are created, (4) a presentation of the inter-relationships among sequences in the form of a general pattern of system operation, (5) the development and encoding of an order structure for a stored program, (6) some basic considerations in the design of circuit logic to implement this order structure, (7) a functional description of the organization of this circuit logic, (8) several examples of how programs are written and (9) brief discussions of the processes of program assembly and system simulation.

XI. ACKNOWLEDGMENT

A large number of people share responsibility for the philosophies, techniques and results described in this article and, in writing it, the authors have tied together and related the work of these people. Particular attention is drawn to W. A. Budlong, G. G. Drew and J. A. Harr because of their substantial contributions to the successful application of the stored program concept to telephone switching. The basic method of handling telephone call functions as described by the system sequences in Part One is an outgrowth of engineering studies made by C. E. Brooks and Miss G. E. Markthaler. In addition, the generalized description of the mode of system operation embodied in Fig. 8 is due to H. Ghiron, and some of the programming examples were taken from material prepared for a course of training in programming by J. W. Howard.

REFERENCES

1. Joel, A. E., An Experimental Switching System Using New Electronic Techniques, B.S.T.J., **37,** September 1958, pp. 1091–1124.
2. Feldman, T. and Rieke, J. W., this issue, pp. 1421–1453.
3. Feiner, A. and Goeller, L. F., this issue, pp. 1383–1403.
4. Greenwood, T. S. and Staehler, R. E., A High-Speed Barrier Grid Store, B.S.T.J., **37,** September 1958, pp. 1195–1219.
5. Hoover, C. W., Staehler, R. E. and Ketchledge, R. W., Fundamental Concepts in the Design of the Flying Spot Store, B.S.T.J., **37,** September 1958, pp. 1161–1194.
6. Freimanis, L., this issue, pp. 1405–1419.
7. Yokelson, B. J., Cagle, W. B. and Underwood, M. D., Semiconductor Design Philosophy for the Central Control of an Electronic Switching System, B.S.T.J., **37,** September 1958, pp. 1125–1160.