[illegible]

COPY NO. 283

DATAmatic 1000
AUTOMATIC PROGRAMMING MANUAL
VOLUME I
THE ASSEMBLY PROGRAM

DATAmatic
A Division of Minneapolis-Honeywell Regulator Company
151 Needham Street
Newton Highlands 61, Mass.

Printed in U. S. A.

DSI-6

PREFACE

The present manual represents Volume I of a set of Automatic Programming Manuals. It serves to introduce the concept of automatic programming as applied to the DATAmatic 1000 Electronic Data-Processing System. The main body of this volume is devoted to a description and explanation of an Assembly Program for use with this system. The DATAmatic 1000 body of instructions is reviewed, special Assembly Program instructions are described, and the procedure for writing a program to be assembled is developed, step by step. For the benefit of readers not familiar with the DATAmatic 1000, a brief description of the system precedes the manual.

Volume II is devoted to the DATAmatic ABC-1 Automatic Business Compiler, which permits the programmer to write complicated programs in easily learned codes. This volume also describes the Library Additions and Maintenance Program (LAMP), by means of which the programmer may utilize, modify, and/or add to a set of frequently used routines stored on a special tape called the Subroutine Library. This Subroutine Library is listed and described in a loose-leaf appendix to the Automatic Programming Manuals.

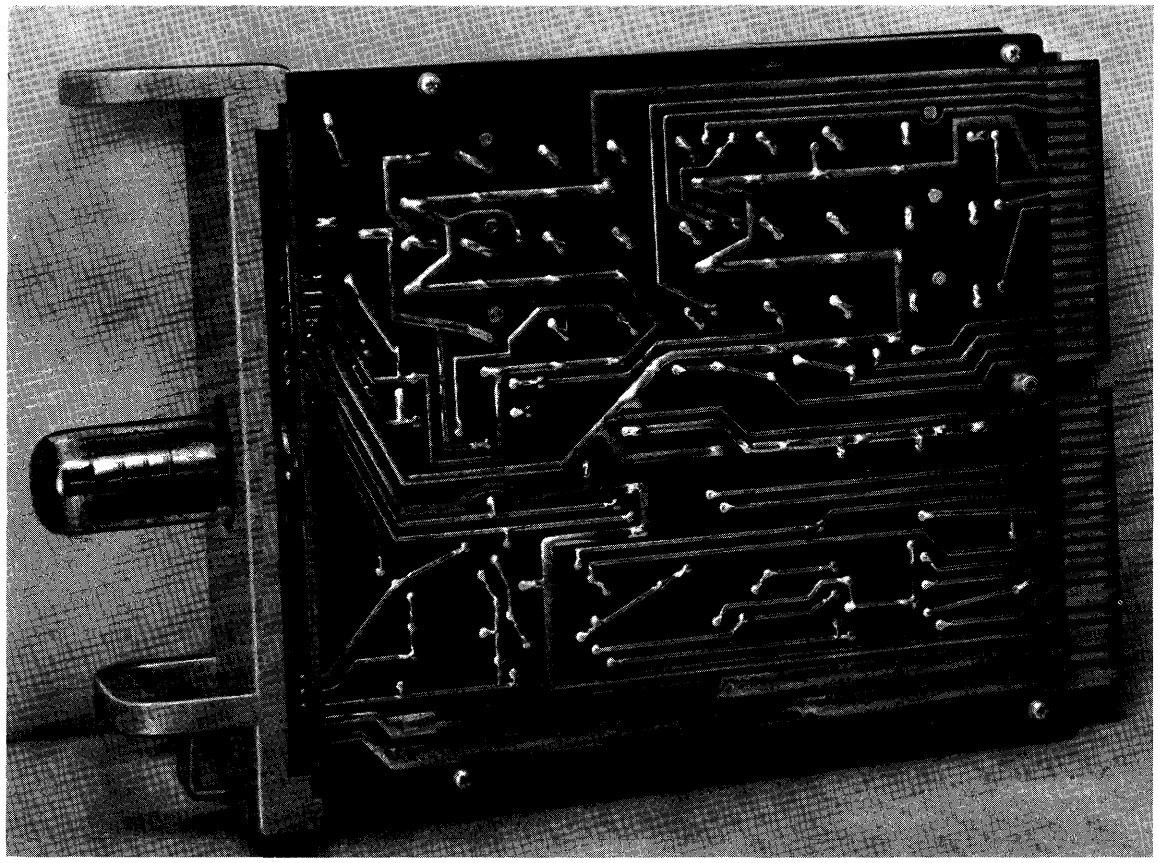
Volume III is a Utility Manual which describes a number of Service Routines, such as a Tracer Routine, a Storage Print Routine, a Program Modifier Routine, and a Tape Editor Routine. These routines perform service functions which facilitate maintenance and use of the various automatic programming devices available with the DATAmatic 1000.

TABLE OF CONTENTS

The DATAmatic 1000 is a high-capacity electronic data-processing system designed specifically for application to the increasingly complex problems and procedures required in modern business. The system incorporates significant new systems techniques as well as several basically new component developments. One of the primary features of the DATAmatic 1000 is its exceptionally large capacity to store information on magnetic tape, coupled with its ability to feed information from magnetic tape to the processing section and back to magnetic tape at a sustained rate of 60,000 decimal digits per second. In addition, the operational speed of the processing section maintains full compatibility with this high speed of information transfer.

Two of the most cumbersome aspects of most business problems are sorting and file maintenance. The DATAmatic 1000 is equipped with an extensive and flexible set of instructions, designed specifically to excel in the performance of these functions and many others. These instructions may be automatically assembled into complete programs by the DATAmatic ABC-1 Automatic Business Compiler. Thereafter, a task which is repeated daily or weekly is initiated simply by reusing the program from its storage on the program magnetic tape.

In the DATAmatic 1000, reliability is a prime consideration throughout every aspect of engineering and design. The design of electronic circuitry is highly conservative. Every transfer of information within the system is carefully checked to insure that the information is transferred without alteration. In addition, all arithmetic and logical operations are completely checked. All units of the system are constructed of easily replaced standard packages to facilitate maintenance. A system of marginal checking includes circuitry and a special program which may be run periodically to locate any package which should be replaced because of marginal performance. With proper use of this facility, most machine malfunctions will be corrected before they occur.

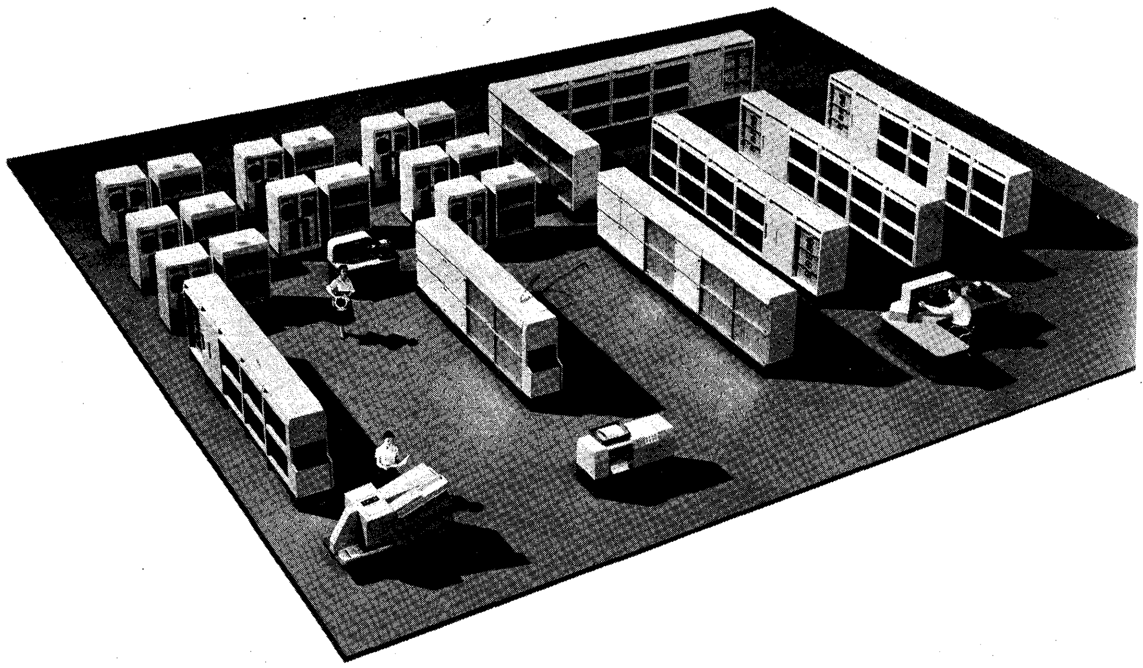


A High-Speed Memory Amplifier package representative of the modular construction used throughout the DATAmatic 1000 system

Elements of the System

The system may be conceived functionally as comprising three main sections, the Input, Central Processor, and Output Sections, although its physical layout will generally not correspond with such a conception. Data is initially fed into the Input Section in the form of punched cards. This section, which includes a Card Reader, an Input Converter, and one Magnetic File Unit, reads the data from the cards, translates it into machine language, edits and arranges it into the desired format, and records it on magnetic tape.

The Central Processing Section includes (1) Arithmetic and Control Units, (2) the High-Speed magnetic-core Memory, (3) Magnetic File Units, (4) Input and Output temporary storage Buffers, and (5) the



Typical Layout of a DATAmatic 1000 Electronic Data-Processing System

Central Console. The Central Processor reads data stored permanently on magnetic tape, performs all manipulations of data, controls the sequence of functions performed, stores information temporarily while it is being processed and, after processing, stores it permanently on magnetic tape. By means of the Central Console, the operator may monitor the overall operation of the system. As needed, Magnetic File Units may be used by auxiliary equipment. Such action is controlled by switches.

The Output Section converts data from magnetic tape into either punched-card form or printed form, performing considerable editing in the process. The Model 1300 Output Converter, which feeds standard punching and/or printing equipment, may either replace or supplement the Model 1400 High-Speed Output Converter and

Printer, depending on the quantitative requirements of the system for output information. One Magnetic File Unit may be considered a part of the Output Section.

Magnetic Tape Storage

The basic medium for the storage of information in the DATAmatic 1000 is magnetic tape. The particular tape used, the method of recording information on it, and the tape-handling equipment have all been designed or selected to be mutually compatible and to provide high capacity, ease and speed of access to information, ultra-reliable storage and recovery of information, and maximum utilization of space on the tape.

Type VTR-179 magnetic tape has been selected for the DATAmatic 1000 because of its reliability and long life. This tape consists of a layer of iron oxide bonded to a tough, durable Mylar plastic base. A reel of tape is three inches wide and 2700 feet long and can store over 37,000,000 decimal digits of information, the equivalent of data which would require 465,000 punched cards.

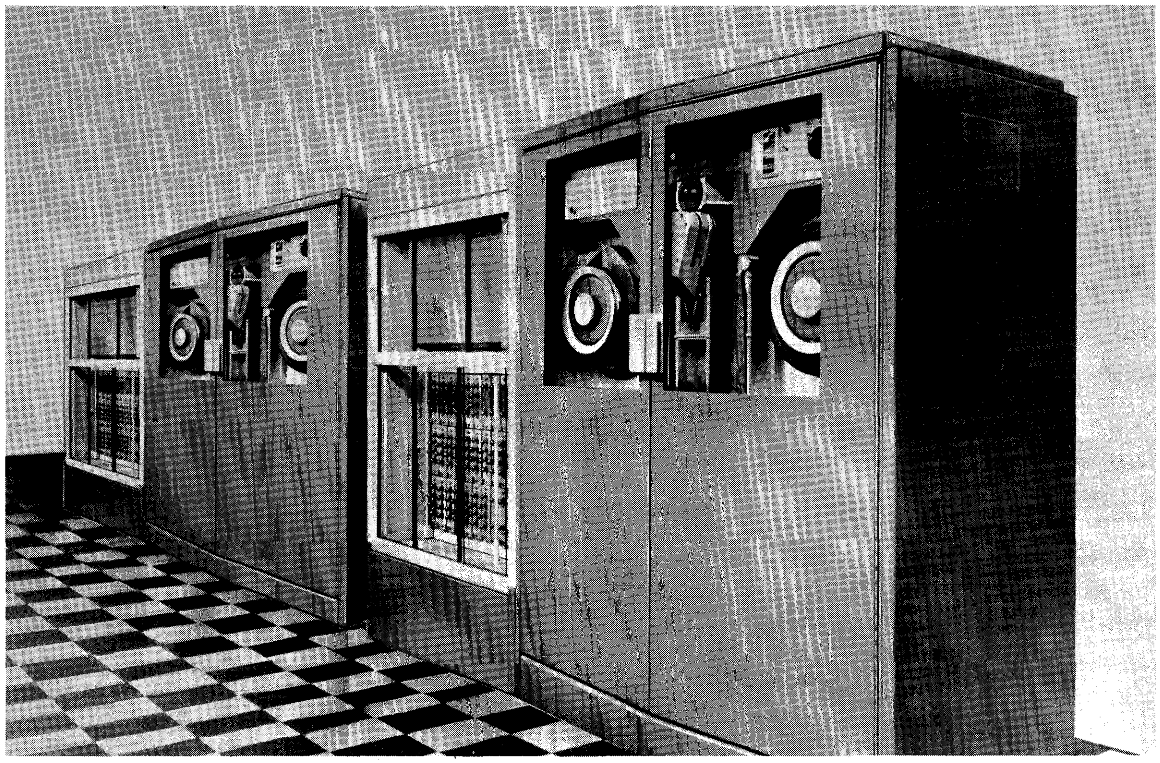
Stored information is recorded on the magnetic tape in groups of magnetized spots. The length rather than the strength of these spots is used to form a dot-dash code representing the encoded digits, letters, and symbols. This, the first of a series of unique reliability features, assures that variations in the recorded signal strength will not result in errors.

The information is stored in standard quantities called blocks, which are arranged in a novel fashion along the tape. The virtual elimination of dead space and the optimum packing of information into the tape area is achieved by regarding the tape as a series of areas one block in length, then recording in every other block while the tape travels in one direction and in the blocks omitted while the tape travels in the reverse direction. The blocks not filled in a given direction of travel provide the space for starting and stopping the tape in that direction. As a result, information is recorded on almost the entire area of the tape. Moreover, since the reversal of tape direction is accomplished automatically, all of this information is written or read sequentially and the tape is positioned at its physical beginning at the conclusion of this process.

Information is recorded lengthwise to the tape in 31 channels, a system which greatly speeds the transfer of information and facilitates searching processes. Specifically, as many as ten tapes may be searched simultaneously, which means that the system is actually passing over 600,000 decimal digits per second while seeking the particular item desired. The read-record head will write on the tape at the sustained rate of 60,000 decimal digits per second and will recover this information at the same rate. The reading or searching operations may be performed with the tape travelling either forward or backward.

The tape-drive mechanism and the read-record head are contained in the Magnetic File Unit. An installation may include from four to one hundred Magnetic File Units, all directly connected into the system. They may be divided in any manner and at any time between the reading and recording operations. The volume of transactions and the complexity of operations govern the number of Magnetic File Units required for a given system. Furthermore, these units may be added to or removed from the system at any time as these requirements vary.

In order that any Magnetic File Unit may be interrogated and information recovered from it without interrupting the operation of the Central Processor, a File Reference Unit is available. Thus a Magnetic File Unit may, at different times, be recording data received from the Input Converter, reading data to the Output Converter, recording data



DATAmatic 1000 Magnetic File Units

from the Central Processor, reading data to the Central Processor, or reading data to a File Reference Unit. Also available is a File Switching Unit which increases the flexibility with which Magnetic File Units may be arranged into the various functional groups.

Input Section

Data enters the DATAmatic 1000 on standard 80-column punched cards which are initially read by the Card Reader. In this unit the card is read twice, the two readings are compared, and the card is stacked. If the two readings of the card are not identical, the operation of the Input Section will stop and the card will be sent to a reject hopper. The Card Reader holds batches of over 3000 cards at one time and passes them at the rate of 900 fully punched cards per minute.

The information which is read from the punched cards is translated into the language of the system and arranged in the format of the magnetic tape by the Input Converter. In this process, it passes through two control panels and two temporary storage locations, providing great flexibility for transposition, duplication, and discarding of information. The operator manually sets an identifying control number into the Input Converter, which includes this number in the information to be written on magnetic tape. The control number may then be written on the batch of cards which it represents, in case it is desired later to cross-reference these cards with their corresponding tape.

The encoded information is first arranged in a 100-column format within the converter. In this conversion, any number of card columns may be duplicated provided that the total number of columns does not exceed 100. Triplication of columns is not permitted. Thirteen conversion rules are available for the translation of punch code into machine code. Any single card column may be translated by any one of these thirteen rules. The information is then translated into the final tape format, the contents of two punched cards being fed to each block on the magnetic tape. Several automatic checking features are built in to detect improperly punched cards or errors either in reading or in one of the conversion steps. The operator controls the settings of a group of panel switches which direct the course of action that the machine is to follow in each of these situations.

It must be emphasized that the operation of the Input Converter is strictly "off-line". That is, it proceeds entirely independently of and simultaneously with the data-processing and/or output functions. Normally, one or more specific Magnetic File Units are assigned the function of writing on tape all raw data received from input and communicating it to the Central Processor.

Binary Notation

Information which is manipulated, stored, or communicated other than by electronic systems is generally written using 10 decimal digits, 26 alphabetic characters, and a number of punctuation marks and other special symbols. Basic to the adaptation of information to electronic systems is the fact that such information can be written entirely in terms of two symbols, generally called zero and one. This presentation is called binary notation and is analogous to the presentation of information in the more familiar Morse Code, in which the two symbols used are called dots and dashes. The symbols used in a binary notation are called binary digits or bits. For example, the ten familiar decimal digits, 0 through 9, are represented in binary notation as follows:

$\bar{0}\bar{0}\bar{0}\bar{0}$	- 0	$\bar{0}\bar{1}\bar{0}\bar{1}$	- 5
$\bar{0}\bar{0}\bar{0}\bar{1}$	- 1	$\bar{0}\bar{1}\bar{1}\bar{0}$	- 6
$\bar{0}\bar{0}\bar{1}\bar{0}$	- 2	$\bar{0}\bar{1}\bar{1}\bar{1}$	- 7
$\bar{0}\bar{0}\bar{1}\bar{1}$	- 3	$\bar{1}\bar{0}\bar{0}\bar{0}$	- 8
$\bar{0}\bar{1}\bar{0}\bar{0}$	- 4	$\bar{1}\bar{0}\bar{0}\bar{1}$	- 9

Bars will sometimes be placed over binary digits when there is some danger of confusing them with decimal zeros and ones.

The storage of information by electronic equipment depends upon the ability to distinguish between two states which represent the two symbols used in binary codes. There are many electronic devices which can make such a distinction. An example of such a device which is both fast and reliable is the tiny, ring-shaped magnetic core. This core may be magnetized in either of two senses; in one sense it is considered to be storing a binary zero and in the other sense a binary one. These tiny magnetic cores constitute the principal element for the storage of information in the High-Speed Memory and buffer storage units of the DATAmatic 1000 Central Processor. In a group of four such magnetic cores, ten of the sixteen possible combinations of states may be used to represent the ten decimal digits.

Central Processing Section

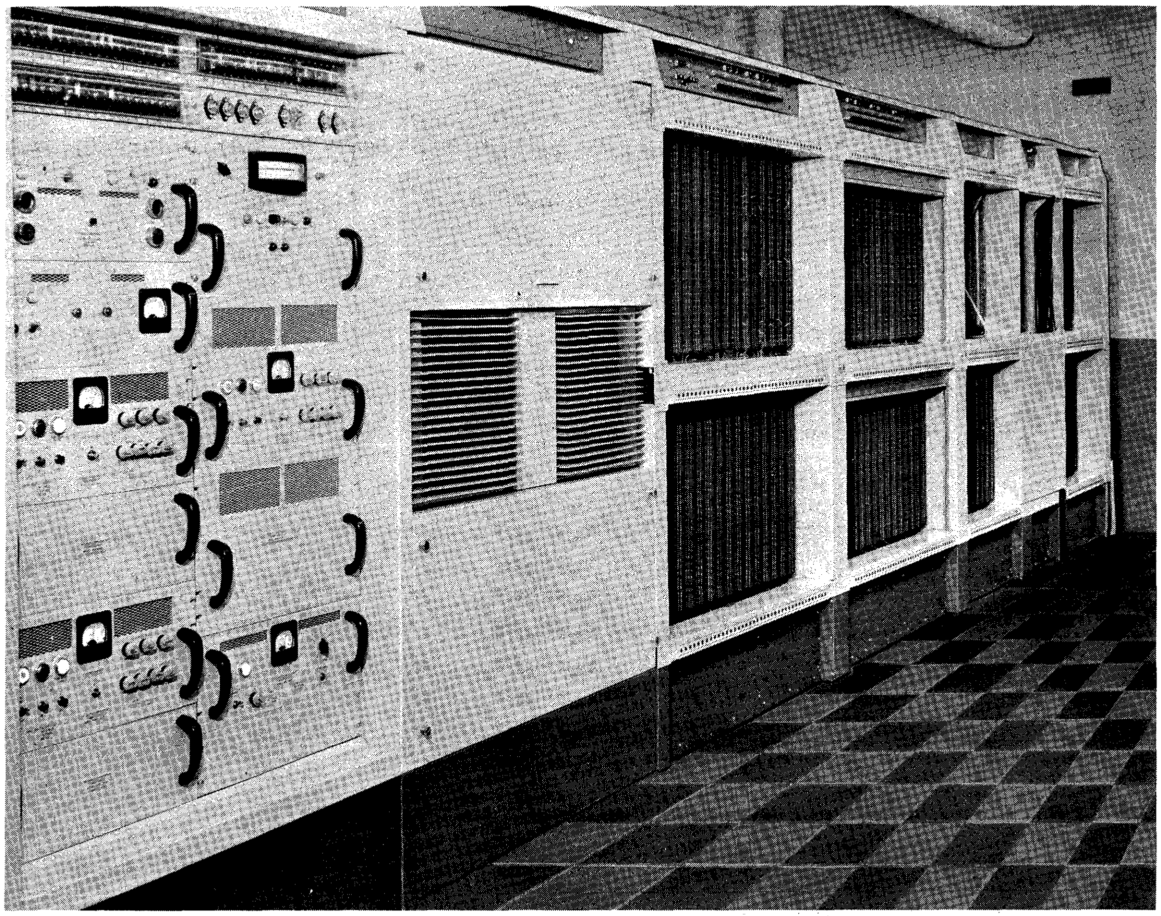
The Central Processor has already been defined to include the Arithmetic and Control Units, the Input and Output Buffer Storage Units, the High-Speed Memory, the Magnetic File Units, and the Central Console. Together, these units contain the electronic elements and circuitry for high-speed performance of the stored programs.

The fast and reliable internal memory is composed of over 100,000 magnetic cores and has a capacity of 24,000 decimal digits. Access is in parallel for rapid readout of stored information. Processing of data stored on magnetic tape is also enhanced by the inclusion of two Input and two Output Buffer Storage Units. These buffers, which are each capable of storing 744 decimal digits, permit a steady flow of information to and from memory and enable the memory to read from one tape and write on another simultaneously.

The design of the Central Processor and the provision of certain special instructions are specifically aimed at the attainment of high sorting, merging, and file-maintenance speeds. Some examples of the speeds achieved are:

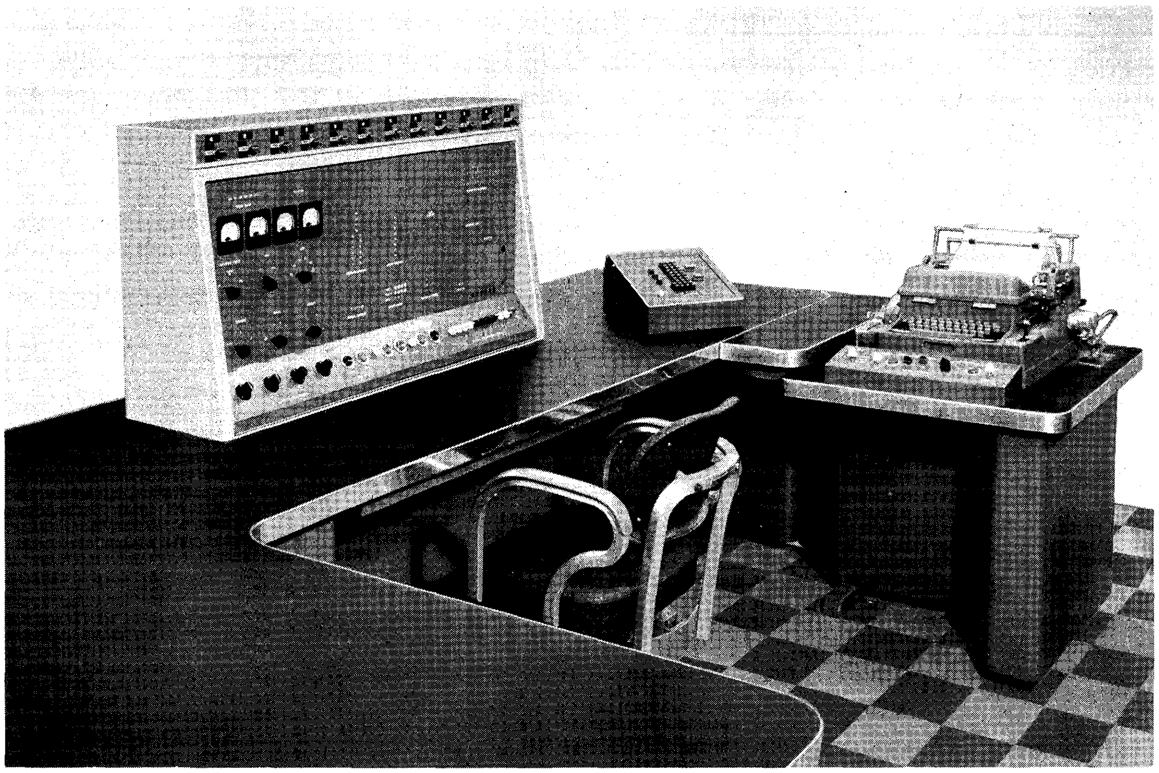
Sort -	60,000 decimal digits per second (equivalent to 750 fully punched cards per second).
Merge -	60,000 decimal digits per second.
File Maintenance -	600,000 decimal digits per second.

Arithmetic instructions are carried out by the Arithmetic Unit. The sequence of performance of the stored instructions is directed by the Control Unit. The Central Console is the means of human communication with, and control over, the system. It affords active human control over starting and stopping the machine and passive communication in displaying a continuous picture of the status of the DATAmatic 1000 as it processes a program instruction by instruction. The latter property is an exceptionally useful diagnostic tool for program debugging. The



High-Speed Memory section of the DATAmatic 1000 Central Processor

Central Console also mounts a special automatic typewriter which is used for the manual insertion of data to the machine and for the interrogation of the machine. Thecomponentsof the system can be checked out by running the marginal check program. The Console displays the results which indicate whether any package in the system is approaching an unsatisfactory level.



DATAmatic 1000 Central Console showing simplicity of layout achieved through functional design

A fundamental reliability feature of the system is the fact that each basic unit of information includes a check digit called the weight count. This weight count is recomputed after each transfer of information within the system. The arithmetic comparison of the original and the recomputed weight counts is an extremely positive and economical means of verifying all internal information transfers, plus arithmetic and logical operations.

Output Section

The Output Section, like the Input Section of the DATAmatic 1000, operates entirely "off-line". It accomplishes the conversion of information stored on magnetic tape into the form of punched cards or printed copy. Two alternative output sections are available which may be purchased either singly or together, depending on the quantity and speed requirements of the application. These are the Model 1300 Output Converter which provides the required output to drive a standard card punch and/or a standard 150-line-a-minute printer, and the Model 1400 High-Speed Output Converter which includes a special DATAmatic High-Speed Printer capable of printing 900 lines per minute. As is the case in the Input Section, one or more Magnetic File Units are normally assigned to communicate between the Central Processor and the Output Converter.

Model 1300: The Model 1300 Output Converter reduces data stored on magnetic tape to a form acceptable to a standard 150-line-per-minute tabulator and/or a standard 100-card-per-minute card punch. The tabulator and card punch functions, governed by standard control panels, are preserved.

Information is read from magnetic tape to the converter in quantities of up to 192 decimal digits, 128 alphabetic characters, or equivalent. Each of these sets of data is processed individually and becomes the basis of one line of printed output and/or one 80-column punched card.

The data is then read into converter output storage through a code translator, controlled by a conversion control panel. There are 14 rules for the translation of machine language into standard punch card code.

The output storage section simulates 120 columns of punch-card data, in which form the information leaves the converter. Format arrangement and all other standard printout functions are governed in normal fashion by the control panels associated with the readout equipment. In the case of the card punch, the data is converted into the standard 80-column format and transposition and duplication of columns are effected, as desired, by proper wiring of the card punch control panel.

Model 1400: The Model 1400 High-Speed Output Converter operates from a completely flexible tape format and performs a considerable amount of editing and format arrangement while preparing information to be printed at the rate of nine hundred 120-character lines per minute. In fact, the most complicated printed formats are obtained with a minimum amount of pre-editing required in the Central Processor. Special symbols and legend material can be emitted. Also the printing of certain parts of the form may be suppressed, dependent upon the contents of other data within the particular record. Furthermore, the same output tape may be used for several different types of printing runs by wiring and using all of the control panels in the equipment. The sequence of information on magnetic tape need not have any relation to the sequence of printing of information within a given line. It is, moreover, possible to scan a record on the tape several times, on each occasion deriving different combinations of data to be printed on a given form; data from the tape may be rejected or printed several times at will.

From the moment that information is read from the magnetic tape to the actual printing process, a complete train of information monitoring exists to preclude the possibility of printing erroneous information. This system includes a read-back signal from the actual printing hammer to the original stored information to verify the correctness of the character being printed in every column of the form.

The High-Speed Output Converter reads information from magnetic tape in discrete quantities of up to 192 decimal digits. These quantities may be read from any part of the block and are handled as separate units of information throughout the conversion process. Three control panels are used to select the input information, translate it, and store it in the 120-position converter storage. There are 160 printing positions available on the High-Speed Printer, of which any

120 may be used during a given run. Two additional control panels are used to select the particular 120 print positions to be used and to perform further editing.

Integrated Checking

The weight count feature of the DATAmatic 1000, previously described, is an integrated checking system which verifies every information transfer, arithmetic and logical operation from the original conversion to machine language through the final production of printed or punched output. The weight count digit is originally computed and checked during the input conversion process. It is then recorded on tape, one such digit being an integral part of each basic unit of information and remaining with this basic unit throughout all of the operations of the system. Thereafter, recomputation and checking of weight count verifies every transfer of information from tape to the Central Processor, and all internal operations within the Central Processor, transfers from the Central Processor to tape, transfers from tape to the Output Converter, and Output Converter decoding processes. Each of these checking sequences is integrated with the preceding and following sequence to form a single, system-wide verification of accuracy. The weight count system is augmented in various DATAmatic 1000 units with duplicate circuitry and other special circuits which further extend the checking system.

Automatic programming routines aid in preparing programs for electronic data-processing systems by replacing many repetitious manual tasks with automatic machine functions. Not only are time and money saved, but programming accuracy is greatly enhanced. In fact, the sheer volume of programming required by large data-processing applications has made such routines a practical necessity. In order to illustrate how such routines assist the programmer, it is necessary first to describe the steps associated with conventional program preparation and then to show the manner in which automatic programming can replace some of these steps or minimize the work associated with them .

Manual Programming Procedure

The preparation of a program to perform a large-scale data-processing operation without the use of automatic programming can be broken down into these eight major steps:

(1) Analysis of the Operation
(2) System Design
(3) Program Design
(4) Coding
(5) Input Preparation
(6) Checkout Planning
(7) Checkout
(8) Program Operation

Step 1. Analysis of the Operation. In a data-processing operation, the machine processes a large quantity and a wide variety of data in order to produce the required information. Therefore, before a method of approach can be considered, the operation to be performed must be carefully analyzed. All of the specific inputs must be designated, the frequency and manner of processing them must be determined, and the volume of each type of input data must be estimated. The same consideration must be given to the required output information. This analysis is frequently made by means of flow charts which show the interconnections and sequential relationships of these inputs, processes, and outputs.

Step 2. System Design. With the inputs, processes, and outputs of the operation clearly defined, it is possible to design a programming system. The initial task at this step is to specify the format of the data being processed at the input stage, the processing stages, and the output stage, i. e., the way in which the information will be punched on input cards, the format of the information on magnetic tape files, and the printed or punched output format. The procedure for operating the data-processing system must also be considered; tape changing during program operation, console operation, control panel wiring, and control information printed out at the operator's console all play an important role in the design of the system.

At this point, the preparation of the actual program begins. A block diagram is prepared which shows the logical steps that the input data must go through in order to produce the required output information and the sequence in which these steps are to be performed. This diagram consists of a series of blocks, one for each logical step, with lines connecting the blocks to indicate the sequence in which they are performed.

Step 3. Program Design. Each block of the block diagram is next broken down into a number of boxes, each of which specifies a function to be implemented by a few machine instructions. This new diagram gives a complete picture of how the program will operate on the machine. It is called a programming flow diagram and it provides the link between the flow chart, the block diagram, and the actual program instructions.

Step 4. Coding. The programmer may now begin the coding process, that is, writing the instructions which will direct the data-processing system to perform the functions indicated in each box of the programming flow diagram. Memory locations must be assigned to all program instructions and other data. However, prior to the actual start of coding, the method of operation indicated by the block and flow diagrams should be evaluated and any changes which will improve the program should be introduced at this point.

Step 5. Input Preparation. The coded program must be transcribed onto a medium which the data-processing system can read and translated into a language which it can understand. This function is accomplished by keypunch operators who transfer the information from the programmers' coding sheets onto standard 80-column punched cards. The Input Converter then writes the punched information on magnetic tape.

Step 6. Checkout Planning. Up to this point, the programming system has been gradually broken down into a large number of steps, each of which is implemented by a few machine instructions. These instructions have been assembled in the proper sequence to perform the data-processing operation. After a program has been coded, steps must be taken to verify that it performs the desired functions. First, an overall checkout plan must be prepared; then the steps in the checkout process must be specified in detail.

Step 7. Checkout. The first part of the checkout process is to operate the program using specific controls prepared during step 6 as a part of the overall checkout plan. These controls permit the programmer to examine information at the various logical breaks in the program. As errors are detected in this process, corrections to the coding must be made as specified in step 4 (coding), and step 5 (input preparation) must be repeated. The program must be tested with a variety of input data and it should be made to produce samples of all of the types of output information. Therefore, to conclude the checkout process, a simulation of the entire programming system must be performed on the machine.

Step 8. Program Operation. Each checked-out program requires operating instructions, scheduling information, and set-up plans in order to make the most efficient use of the machine. These instructions must specify the techniques necessary to get the input data into the machine, control the processing of the data, and prepare the required output information. They should also specify all of the special indications which are produced at the operator's console to increase the efficiency of program operation. To complete the documentation, a flow diagram and an annotated copy of the program must be prepared.

Elements of Automatic Programming

ASSEMBLY PROGRAMS: The purpose of automatic programming is to replace the routine portions of these eight steps with machine operations. First of all, the language which the programmer uses differs from the language of the machine. Programmers work best with easily-remembered or mnemonic operation codes (e. g., ADD, SUB, MUL) and decimal numbers; electronic data-processing systems work with binary information. The translation from the mnemonic-decimal language of the programmer into the binary language of the machine is just the sort of task which high-speed data-processing systems do well. Not only is translation accomplished rapidly, but the results are error-free. Programs which perform this operation are usually called Assembly Programs.

COMPILERS: In the program design step, the blocks of the block diagram are broken down into boxes of a flow diagram to be transcribed into machine coding and checked out. Frequently, the same block appears in several programs. Using manual programming procedures, the coding and checkout steps must be repeated each time this block appears. However, much time can be saved in the flow diagramming, coding, and checkout of routines with identical blocks by having the person who first codes the block perform a few extra tasks to preserve the coding for other programmers wishing to use it. Such reusable blocks are called subroutines. The routine task of duplication of coding can be eliminated by extending the Assembly Program so that subroutines can be added to any program by a single instruction. Therefore, by introducing subroutines into the programming system, all of the steps after System Design are effectively and automatically eliminated with respect to the subroutines. A program which is capable of processing subroutines in this fashion is called a Compiler because it compiles completely coded and checked-out subroutines into a program.

SUBROUTINE LIBRARY MAINTENANCE: Subroutines, together with instructions for their use, are stored on magnetic tape or on punched cards in what is called a Subroutine Library. A Compiler may be supplemented by a program which automatically adds, deletes, or modifies subroutines in the Subroutine Library. This Library Maintenance Program, as it is generally called, relieves the programmer of another routine task.

RELATIVE AND SYMBOLIC CODING: Assembly Programs and Compilers frequently utilize either relative or symbolic tags which permit a programmer to refer to instructions and data in his program without having to assign them specific memory locations. Words with relative tags are coded in groups and have definite relative positions within these groups. After detailed coding is completed, the programmer assigns memory areas to these groups of relative tags. Words with symbolic tags are automatically assigned memory locations by the Compiler or Assembly Program. Both systems simplify detailed coding and also enable programmers to make program modifications, additions, and deletions without extensive recoding.

UTILITY ROUTINES: The automatic routines which have been described assist in the preparation of programs. There is another group of automatic routines which aid in the checkout phase of program preparation. These are called Utility Routines.

In the checkout step, the programmer is assigned short periods of time on the machine. During these periods, he must operate his program and obtain sufficient information to locate and analyze any errors it may contain. A listing of the contents of memory at various stages of this process may provide the necessary information. The program which produces such a listing is called a Post Mortem or Memory Dump Routine. These routines convert the contents of memory from the binary form to the mnemonic and decimal language of the programmer.

Some types of errors are difficult to track down using the Post Mortem Routine. Under such conditions the programmer would like to know exactly what happens as each instruction is performed in the region of the error. This could be accomplished manually by using the Central Console to examine the affected memory locations after performing each instruction. However, this process is extremely wasteful of machine time. A program called a Tracing Routine performs this function automatically at high speeds. The Tracing Routine produces a listing of the instructions performed in the sequence in which they were performed and for each instruction specifies the contents of the affected memory locations.

Data-processing systems work with information stored on magnetic tape; therefore, certain procedures are necessary to insure efficient handling of the data. For example, routines which will copy files from one tape to another, locate, write, and modify files, and edit tape files for printing must be available as a part of an automatic programming system.

There are many other programs in an automatic programming system, some which perform simpler functions and some which are much more sophisticated. The complete set of automatic programs provided for DATAmatic 1000 customers, the DATAmatic Automatic Business Compiler System (ABC-1), is described in this and the following volumes.

SECTION I - DATAmatic 1000 ASSEMBLY PROGRAM

The Assembly Program is a routine which translates programs from the alphabetic and decimal language of the programmer to the binary language of the computer. The programmer writes the instructions of his program using three-character mnemonic operation codes. Each word being coded is assigned a tag which indicates its location in memory and which may be used within instructions to refer to the word. A tag may be the absolute address of a specific location in the High-Speed Memory or it may be a relative tag which designates an address in relation to a specific memory location. Control information required by certain instructions, i. e., number of shifts, number of words transferred, or tape identification codes, is specified in decimal numbers. The Assembly Program processes programs written in this form, translating the mnemonic operation codes to the proper internal codes, assigning memory locations to relative tags, converting decimal numbers to the appropriate binary configurations, and producing a copy of the program in machine language on magnetic tape.

This machine language form of the program is written on magnetic tape in a prescribed format. This format contains self-loading control words which automatically read the program into High-Speed Memory upon request after the read-in process is initiated.

SECTION I - THE ASSEMBLY PROGRAM

PROGRAMMER'S LANGUAGE

Table I, pages 2-5, contains a list of the symbolic instructions which may be used with the Assembly Routine. All DATAmatic 1000 instructions are included in this list. The complete specifications for the instructions are given in Section I of the DATAmatic 1000 Programming Manual. There are special instructions for interpreting three kinds of constants: signed numeric words consisting of a plus or minus sign and eleven hexadecimal digits; unsigned numeric words containing twelve hexadecimal digits; and alphanumeric words containing eight Latin letters, decimal digits, or special symbols.

Table II, page 7, contains a list of 6-bit interpretations of all codes used as input to the Assembly Program. These include letters, numbers, hexadecimal digits, and special symbols. When a 4-bit interpretation of a code is required, as in numeric words and instructions, the Assembly Program makes the necessary conversion.

Word Structure

The DATAmatic 1000 word structure is as follows:

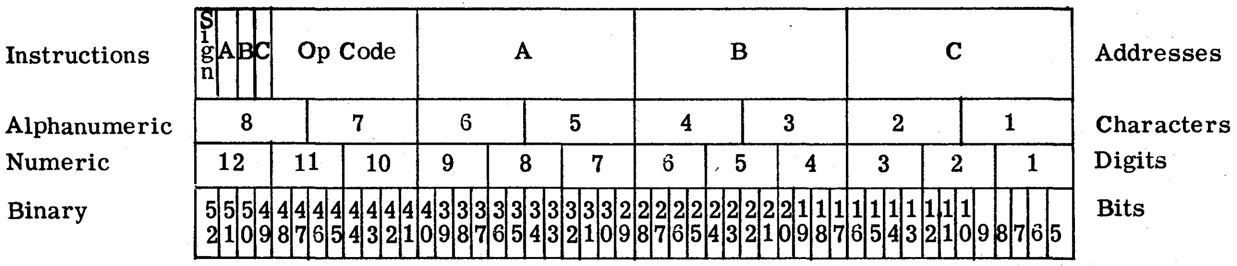


Figure 1

Reference to the actual DATAmatic 1000 word structure uses the notation shown in Figure 1. For example, the terms Character Position 3, Digit Position 7, Bit Position 25, or Address A are used (Address A consists of the bank designator in bit position 51 and the subaddress in bit positions 40 to 29).

2 PROGRAMMER'S LANGUAGE

SUMMARY OF DATAmatic 1000 INSTRUCTIONS

INSTRUCTION TYPE	INSTRUCTION FORM	FUNCTION
Add	ADD/X/Y/Z □	Add the contents of X to the contents of Y and store in Z. X and Y remain unchanged. Subsequence to 1988 in case of overflow.
Subtract	SUB/X/Y/Z □	Subtract the contents of Y from the contents of X and store in Z. X and Y remain unchanged. Subsequence to 1988 in case of overflow.
Multiply	MUL/X/Y/Z □	Multiply the contents of X by the contents of Y and store the rounded high-order product in Z. Store the low-order product with 5 added to the high-digit position in 1995.
Divide	DIV/X/Y/Z □	Divide the contents of Y by the contents of X and store in Z. The remainder is stored in 1995.
Shift Left (preserving sign)	SLP/X/nN/Z □	Shift the contents of X, n numeric places left and store in Z. Sign character and the contents of X remain unchanged.
Shift Right (preserving sign)	SRP/X/nN/Z □	Shift the contents of X, n numeric places right and store in Z. Sign character and the contents of X remain unchanged.
Shift Left (with sign)	SLW/X/n$\begin{Bmatrix}A\\N\\T\end{Bmatrix}$/Z □	Shift the contents of X, nN (n Numeric), nA (n Alphabetic), or nT (n Two-bit) places to the left and store in Z. Sign character is included in the shift. Contents of X remain unchanged.
Shift Right (with sign)	SRW/X/n$\begin{Bmatrix}A\\N\\T\end{Bmatrix}$/Z □	Shift the contents of X, nN (n Numeric), nA (n Alphabetic), or nT (n Two-bit) places to the right and store in Z. Sign character is included in the shift. Contents of X remain unchanged.
Substitute	SST/X/Y/Z □	Insert into Z, the bits of (X) for which there is a 1 in the corresponding bit positions of (Y). Leave unchanged the bits of (Z) for which there is a 0 in the corresponding bit positions of (Y).
Transfer In (A Buffer) (B Buffer) (Same Buffer) (Different Buffer)	TIA/X/N/S □ TIB/X/N/S □ TIS/X/N/S □ TID/X/N/S □	Transfer N words from the specified buffer section into N consecutive HSM locations beginning at location X. Subsequence to S.
Transfer In Bypass (A Buffer) (B Buffer) (Same Buffer) (Different Buffer)	TBA/X/N/S □ TBB/X/N/S □ TBS/X/N/S □ TBD/X/N/S □	Bypassing the interlock, transfer N words from the specified buffer section into N consecutive HSM locations beginning at location X. Subsequence to S.
Double Transfer and Select (A Buffer) (B Buffer) (Same Buffer) (Different Buffer)	DTA/X/N/m/S □ DTB/X/N/m/S □ DTS/X/N/m/S □ DTD/X/N/m/S □	1) Transfer N words to Output Buffer from consecutive HSM locations beginning at X. 2) Replace these N words by N words from the specified Input Buffer Section. 3) Use the mth word and the Extractor Register to form the selection digit. Add digit to units digit of S and Subsequence to the modified address. 4) Store the instruction in Select Order Register (1994).

For complete details of DATAmatic 1000 instructions, see the Programming Manual, Sec. I

Table I (Page 1)

INSTRUCTION TYPE	INSTRUCTION FORM	FUNCTION
Double Transfer and Select--Bypass (A Buffer) (B Buffer) (Same Buffer) (Different Buffer)	 DBA/X/N/m/S □ DBB/X/N/m/S □ DBS/X/N/m/S □ DBD/X/N/m/S □	Same as Double Transfer and Select except that the interlock is bypassed.
Transfer and Select (A Buffer) (B Buffer) (Same Buffer) (Different Buffer)	 TSA/X/N/m/S □ TSB/X/N/m/S □ TSS/X/N/m/S □ TSD/X/N/m/S □	Same as Double Transfer and Select except that step 1, the transfer of words to the Output Buffer, is omitted.
Transfer and Select --Bypass (A Buffer) (B Buffer) (Same Buffer) (Different Buffer)	 BSA/X/N/m/S □ BSB/X/N/m/S □ BSS/X/N/m/S □ BSD/X/N/m/S □	Same as Transfer and Select except that the interlock is bypassed.
Internal Select	ISL/X/Y/S □	Use X and Y to form a selection digit. Add this digit to units digit of S and Subsequence to the modified address.
Transfer Out	TXO/X/N/S □	Transfer N words from consecutive HSM locations beginning at X to the Output Buffer. Subsequence to S.
Transfer Internally	TXI/X/Z/N □	Transfer N words from consecutive HSM locations beginning at X to consecutive HSM locations beginning at Z.
Twin Transfer	TTX/X/Y/Z □	Transfer the word in the Select Order Register (1994) to X and the word at Y to Z.
Transfer and Subsequence Call	TXS/X/Z/S □	Transfer the contents of X to Z. Subsequence to S.

Read Forward	RFA/T/C/S □
	RFB/T/C/S □
	RFD/T/C/S □
		Read next block in forward direction on Tape T into the
		A = A Buffer
		B = B Buffer
		D = Different Buffer.
		Change Sequence Register to C. Subsequence to S.

Read Backward	RBA/T/C/S □
	RBB/T/C/S □
	RBD/T/C/S □
		Read next block in backward direction on Tape T into the
		A = A Buffer
		B = B Buffer
		D = Different Buffer.
		Change Sequence Register to C. Subsequence to S.

Read Forward, Key Channel	RFK/T/C/S □	Put key channel of Tape T in read state and satellite channels in write state. Read Key Channel of next block in forward direction into B Buffer. Change Sequence Register to C, Subsequence to S.
Read Backward, Key Channel	RBK/T/C/S □	Same as Read Forward, Key Channel except that the tape is moved backward.
Print Alphabetic	PRA/X/C/S □	Print contents of X as 8 alphanumeric characters on the Console Typewriter. Change Sequence Register to C. Subsequence to S.

Table I (Page 2)

INSTRUCTION TYPE	INSTRUCTION FORM	FUNCTION
Print Numeric	PRN/X/C/S □	Print contents of X as 12 hexadecimal characters on Console Typewriter. Change Sequence Register to C. Subsequence to S.
Write Forward	WFA/T/C/S □	Put key and satellite channels in write state. Write one block on Tape T. Change Sequence Register to C. Subsequence to S.
Write Forward, except Key Channel	WFP/T/C/S □	Put key channel in read state and satellite channels in write state. Write one block. Change Sequence Register to C. Subsequence to S.
Search Forward, Reading	SFR/T/C/S □	Put key and satellite channels in read state. Search one block forward. Change Sequence Register to C. Subsequence to S.
Search Backward, Reading	SBR/T/C/S □	Same as in Search Forward, Reading except that tape moves in backward direction.
Search Forward, Writing	SFW/T/C/S □	Put key channel in read state and satellite channels in write state. Search forward one block. Change Sequence Register to C. Subsequence to S.
Search Backward, Writing	SBW/T/C/S □	Same as in Search Forward, Writing except that tape moves backward.
Rewind Tape	REW/T/C/S □	Rewind Tape T. Change Sequence Register to C. Subsequence to S.
Switch to First Half	SWF/T/C/S □	Position tape T to read from the first logical half of the tape. Change Sequence Register to C. Subsequence to S.
Switch to Second Half	SWS/T/C/S □	Same as SWF except tape T is positioned to read from the second logical half.
Less Than Comparison, Numeric	LCN/X/Y/C □	If contents of X is numerically less than or equal to contents of Y, change Sequence Register to C.
Inequality Comparison, Numeric	ICN/X/Y/C □	If contents of X is not numerically equal to contents of Y, change Sequence Register to C.
Less Than Comparison, Alphabetic	LCA/X/Y/C □	If contents of X is alphabetically less than or equal to contents of Y, change Sequence Register to C.
Inequality Comparison, Alphabetic	ICA/X/Y/C □	If contents of X is not identical to contents of Y, change Sequence Register to C.
First Key Comparison	FKC/X/Y/C □	Transfer contents of first key to X. If contents of X is ~~less~~ Greater than or equal to contents of Y, change Sequence Register to C. Tape number is units digit of X.
Second Key Comparison	SKC/X/Y/C □	Transfer contents of second key to X. If contents of X is less than or equal to contents of Y, change Sequence Register to C. Tape number is units digit of X.
Pass	PSS □	Go to Sequence Register for next instruction.
Sequence Change	SCS/i/C/S □	Change the Sequence Register to C. Subsequence to S. i may contain up to three numeric digits.

Table I (Page 3)

INSTRUCTION TYPE	INSTRUCTION FORM	FUNCTION
Branch and Return	BAR/X/C/S ▭	Store in X an instruction to change the Sequence Register to its present reading. Subsequence to S. Change the Sequence Register to C.
Stop	STOP/i/S ▭	Unconditional stop. Subsequence to S. i may contain up to three numeric digits.
Optional Stop	OST/i/d/S ▭	~~Stop if specified Breakpoint Switch is ON.~~ Subsequence to S. i may contain up to 3 numeric digits.

Stop if Break-point switch specified by d is on. Stop uncond. if d=S ~~stop~~ proceed if d=9.

SUMMARY OF DATAmatic 1000 CONSTANTS

CONSTANT TYPE	CONSTANT FORM	FUNCTION
Alphanumeric Constant	A/Constant ▭	Assemble as alphanumeric constant with the leftmost character in the leftmost position of the word.
Unsigned Numeric	U/u/Constant ▭	Assemble as unsigned numeric constant with the digit position to the left of the decimal point placed in the uth digit position of the word.
Signed Numeric	N/n/Constant ▭	Assemble as a signed numeric constant with the digit position to the left of the decimal point placed in the nth digit position of the word. If no sign appears, a + sign will be inserted.

SUMMARY OF ASSEMBLY PROGRAM CONTROL INSTRUCTIONS

INSTRUCTION TYPE	INSTRUCTION FORM	FUNCTION
Start	START/i/s/Y ▭ see page 16	i is Program Search Code, a maximum of 8 alphanumeric characters; s is Segment Number, 2 digits; Y is starting location of program in absolute.
Segment Start	SEGMENT START/s ▭	s is Segment Number, 2 digits.
Read Directed Segment	RDS/$\begin{Bmatrix}B\\F\end{Bmatrix}$/T/s/Y ▭	B or F determines read direction on Program Tape. T is tape drive of Program Tape. s is Segment Number, 2 digits. Y is Starting Location after segment has been read in.
Tag Assignment	TAG/X/n ▭	Defines the stem, X, by giving it a value, n. This value will be added to the decimal digits following X when it is used. X may have the form RS or R.

Table I (Page 4)

The basic structure of a word in the language of the Assembly Routine is:
Op Code/Zone 1/Zone 2/....../Last Zone □ , where

(1) Op code is the mnemonic code used to specify the machine command, i.e., ADD, SUB, TIS, etc.,
(2) Zone 1......Last Zone are the parameters necessary to specify the complete instruction,
(3) and□is a special symbol used to denote the end of a word.

For example, TIB/1300/16/1800□instructs the computer to transfer in 16 words from the B Buffer and store them in consecutive locations beginning with location 1300, then make a subsequence call to 1800. All zones are decimal, even those which are binary in machine language. The Assembly Program makes the conversion.

<u>Tags</u>

Each word is "tagged" to indicate its location in memory. The tag of a word is called an Identification Tag. <u>Every</u> word has an identification tag. A tag used within a zone to refer to another word is called a Zone Tag. The distinction applies only to the use of the tag; the same tag is an identification tag for the word it labels and a zone tag in the instruction which refers to the labelled word.

Any identification or zone tag referring to a location in memory contains four alphanumeric characters. The two rightmost characters of a tag must be decimal (00-99). The type of alphanumeric characters in the first two positions of a tag determine whether the address is absolute or relative. Both types of tags may be used in the same program and any number of combinations of acceptable tags may be used.

ABSOLUTE TAGS: When a tag is in the range 0001-1999, it is absolute, i.e., it is the address interpreted by the machine. In this case, the programmer codes instructions, such as ADD/1208/1391/0324 □ .

INTERNAL ALPHANUMERIC CODES

Character	Code	Character	Code	Character	Code
'	001010	0	000000	J	100001
#	001011	1	000001	K	100010
@	001100	2	000010	L	100011
+	001101	3	000011	M	100100
=	101010	4	000100	N	100101
$	101011	5	000101	O	100110
*	101100	6	000110	P	100111
¢	101101	7	000111	Q	101000
CR	011010	8	001000	R	101001
.	011011	9	001001	S	110010
□	011100	A	010001	T	110011
(	011101	B	010010	U	110100
)	011110	C	010011	V	110101
:	111010	D	010100	W	110110
,	111011	E	010101	X	110111
%	111100	F	010110	Y	111000
1/2	111101	G	010111	Z	111001
Space	010000	H	011000	/	110001
-	100000	I	011001	&	110000

Note: For the hexadecimal digits 10-15 (to be used in constants only) use the letters B-G, respectively.

Table II

Addresses 1980-1999 are special addresses whose functions are described in Table III, pages 10 - 11. Address 1978 has a special function described under "Starting the Assembly Process," page 26. In absolute coding, the non-significant digits of an address need not be written. Therefore, 492 is equivalent to 0492, 39 is equivalent to 0039. If a zone is all zeros, no digits need be written in it. Figure 2 shows some examples of absolute coding.

Absolute Identification Tags	Absolute Zone Tags
1001	SCS//1542/1593 □
1002	TXS/1394/492 □
1003	BAR/156/39 □
1004	WFA/9 □

This is equivalent to

Absolute Identification Tags	Absolute Zone Tags
1001	SCS/0000/1542/1593 □
1002	TXS/1394/0492/0000 □
1003	BAR/0156/0039/0000 □
1004	WFA/0009/0000/0000 □

Figure 2

There are two types of **RELATIVE TAGS:**

(1) The first type has the form **Rddd**, where **R** is a digit greater than one or a letter which identifies a relative counter, and each d is a decimal digit. These tags permit as many as 1000 words to be included within a single relative counter. All tags of this type having the same first digit are related to the same address.

(2) The second tag form is **RSdd**, in which **R** can be any letter or number, **S** must be a letter, and each d is a decimal digit. This tag form permits up to 100 words to be included within

each relative counter. All tags having the same first two characters are related.

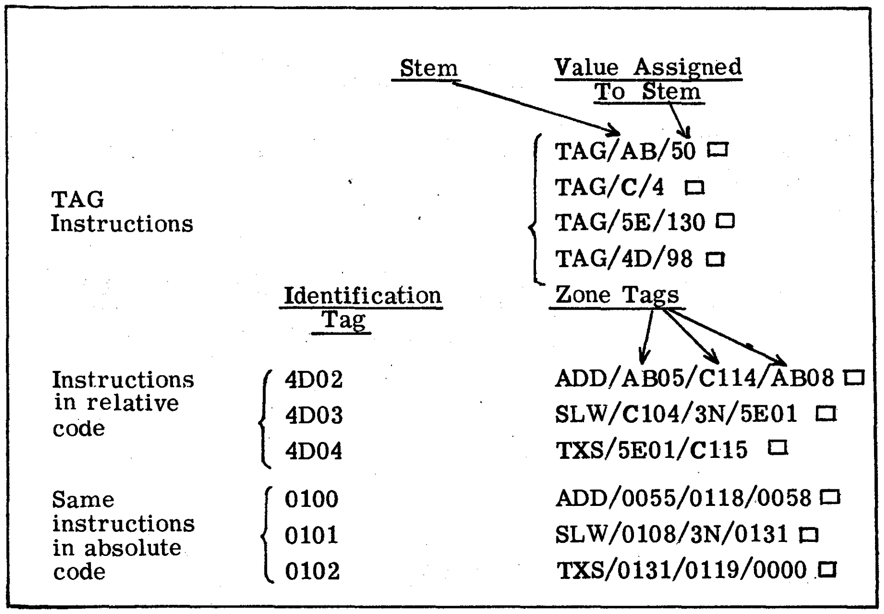
In relative coding, the four digits of an address must be written. The R or RS portion of a tag is called its stem. Before a relative tag is used, the programmer must write a TAG/X/n □ instruction which defines the stem, X, by giving it a value, n. The value of a stem is added to the decimal digits which follow it when used in the program. If, for some reason, the same stem is defined more than once in a program, each definition given will be used by the Assembly Program until the next one appears. Otherwise, the value of a stem remains fixed throughout the entire program. If the stem is not defined at all before a tag using the stem appears, a

TAG
ILLEGAL

error will be printed on the Console Typewriter (see Table IV, pages 30 - 31). If the address which results from adding the decimal digits of a tag to the value of the stem is greater than 1999, a

TAG
ILLEGAL

error will be printed. Figure 3 shows some examples of relative coding.



```
                                        TAG/AB/50 □
                                        TAG/C/4 □
                                        TAG/5E/130 □
                                        TAG/4D/98 □

                 4D02                   ADD/AB05/C114/AB08 □
                 4D03                   SLW/C104/3N/5E01 □
                 4D04                   TXS/5E01/C115 □

                 0100                   ADD/0055/0118/0058 □
                 0101                   SLW/0108/3N/0131 □
                 0102                   TXS/0131/0119/0000 □
```

Figure 3

ADDRESSES OF SIGNIFICANCE

do not use 1972 - 1975 if read in is immediately following assembly

There are several Addresses of Significance in High-Speed Memory, whose function and use are summarized in this chart. Addresses 1981 through 1989 may be used for the following control purposes.

do not use 1984, 85, 87

Address	Possible Control Uses
1981	During the editing pass for data recorded on tape by the Input Converter, if the converter has made an error during conversion, a sentinel subsequence call to 1981 is stored in the 16th word of the item.
1982	The Console has a special start subsequence to 1982. This causes the computer to subsequence to 1982 without altering the Sequence Register.
1983	Illegal punches present during the input conversion process will cause a sentinel instruction with 1983 as the C address to be stored in the 16th word of the item.
✓ 1984	Fillers transferred from the Output Buffer will be sentinel instructions with 1984 as the C address.
✓ 1985	Fillers transferred from the Input Buffer will be sentinel instructions with 1985 as the C address.
1986	Division overflow results in an automatic subsequence call to 1986.
✓ 1987	If the Console Rerun switch is "on" and there is an error in transfer of data out of an input buffer, an automatic subsequence call to 1987 is produced.
1988	Addition or subtraction overflow results in an automatic subsequence call to 1988.
1989	Reaching the end of tape results in an automatic subsequence call to 1989.

Storage Addresses 1990 through 1999, located in the Arithmetic and Control units, are special addresses which store special words used for control purposes. They can be read into and out of by instructions.

Address	Purpose
1990	The Control Register is part of the DATAmatic 1000 control circuitry. It stores each instruction during the time the instruction is being performed by the system.

Address	Purpose
1992 Output Buffer Register	The Output Buffer Register contains the same word as that found in the first word position of the Output Buffer.
1993 Extractor Register	This register contains the constant used for the extraction that is performed in the Transfer and Select instructions.
1994 Select Instruction Register	This register contains one of the operands in the Double Transfer instruction. All Select instructions are stored in this register ~~after~~ before their execution.
1995 Remainder Register	This register receives the remainder after the execution of a Divide instruction. It also receives the low-order product of a multiplication after five has been added to its high-order digit. The low-order product is made up of the eleven low-order decimal digits of the complete product.
1997 Sentinel Register	Any instruction which implements a transfer between the High-Speed Memory and a buffer will also implement the transfer of a word to the Sentinel Register. If the Transfer instruction senses a sentinel, then the first sentinel sensed by the instruction is stored in the Sentinel Register; if no sentinel is sensed, a Pass instruction is stored there.
1999 Current Instruction Register	DATAmatic 1000 instructions are processed in eight word cycles. The Current Instruction Register is used for storing a number of different items of information related to these cycles under different modes of operation. The programmer may use this register as a diagnostic aid when the Central Processor stops.

The address 0000 is also of interest. Except in the case of the Comparison instructions, an instruction to change the Sequence Register to, or make a Subsequence Call to, memory address 0000 (void address) will be ignored and the Sequence Register will be used in the normal fashion. There is a Start at 0000 button on the Console.

TABLE III
Page 2

Note that the TAG instruction has no identification tag. TAG instructions do not get assigned a place in memory. They are only for use by the Assembly Program. It should also be noted that in assigning tags, the programmer must take care to assure proper allocation of the Key Comparison instructions and the Select instructions. In the Key Comparison instructions, the units digit of the A address designates the magnetic tape address code. The value assigned to the stem, therefore, must provide selection of the proper magnetic tape information. The C address of the Select instructions is incremented by the digit generated by the instruction. Since a carry occurs if the sum exceeds nine and an error occurs if it exceeds nineteen, care must also be taken in assigning a place in memory to the word specified by the C address.

Constants

In the discussion that follows, the words "alphanumeric character" are used to denote any of 56 Latin letters, decimal digits, and special symbols which are represented by 6-bit binary codes in the DATAmatic 1000 (see Table II). Similarly, "hexadecimal" is used to denote any of the sixteen characters (ten decimal digits and six special symbols called hex B, C, D, E, F, and G) which represent the sixteen possible groupings of a 4-bit binary code.

There are three methods of specifying constants (see Table I).

(1) Alphanumeric constant: A/Constant□. An alphanumeric constant contains a maximum of eight Latin letters, decimal digits, or special symbols which are translated into 6-bit groups. The leftmost character of the constant is put in the leftmost character position of the word (character position 8) and zeros are inserted as needed. For example, A/BMB41□ is stored as

B	M	B	4	1	0	0	0

The exact bit structure for this word is (see Table II for translation) ,

010010	100100	010010	000100	000001	000000	000000	000000

(2) Signed numeric constant: N/n/Constant □ . A signed numeric constant consists of a plus or minus sign, a maximum of eleven hexadecimal or decimal digits in 4-bit groups, and an indicator (decimal) point. The sign is put in the normal sign position (bits 52-49). If the sign is not written, a plus sign is put in the sign position. The character to the left of the indicator point is put in the "nth" digit position (see Figure 1). If no indicator point appears in the number, the rightmost character of the constant is stored in the "nth" digit position of the word. For example,

from the left.

N/5/ + 207. 56 □
N/7/ - 12454 □

give respectively,

+	0	0	0	0	2	0	7	5	6	0	0

-	1	2	4	5	4	0	0	0	0	0	0

(3) Unsigned numeric constant: U/u/Constant□. The unsigned numeric constant contains a maximum of twelve hexadecimal or decimal digits stored in 4-bit groups and an indicator (decimal) point. The character to the left of the indicator point is put in the "uth" digit position. If no indicator point appears in the number, the rightmost character of the constant is stored in the "uth" digit position of the word (see Figure 1). For example,

U/1/0 □
U/3/B85. 15 □

give, respectively,

0	0	0	0	0	0	0	0	0	0	0	0

0	0	0	0	0	0	0	10	8	5	1	5

Any constant that is neither wholly numeric nor wholly alphabetic must be reduced to one of the three constant forms available by referring to Table II. Any configuration of bits in a word may be expressed as an unsigned numeric constant.

Control Instructions

In order that the Assembly Program produce the program desired, certain control information must be furnished by the programmer. The four Assembly Program control instructions provide the programmer with a method for specifying this information. The TAG instruction, previously described, governs the placing of the operating program in the memory. The START and SEGMENT START instructions serve to identify the beginning of a program or segment, respectively. The READ DIRECTED SEGMENT (RDS) instruction facilitates the automatic interconnection of parts of a program. Detailed descriptions of the control instructions follow.

START/i/s/Y □

- i - Program Search Code
- s - Segment Number
- Y - Starting Location of program in absolute.

The START instruction must be the first card of a program. A card with a modified START instruction must be punched each time the program is reassembled (modified). This card will replace the previous START card. The START instruction is used to set up the File Identification Block on the Program Tape (see Figure 14, page 34).

The program identification or Program Search Code appears in the first zone of the START instruction. The Program Search Code may contain a maximum of eight alphanumeric characters which may be separate and distinct from the Program Identification and Modification Code specified in columns 1-7 on the punched cards (see Figure 5, page 19). However, it is recommended that the 5-character Program Identification followed by the highest Modification Code used on a card in the program be used as the Program Search Code. Each program must have its own unique Program Search Code. It is used to search for the assembled program on the Program Tape. The Program Search Code, the information in columns 1-7 on the card, and the segment number are printed on the Console Typewriter at the start of the assembly of each program.

A segment is a portion of a program. A program may be divided into segments because it is too large to be stored in the High-Speed Memory at one time and/or because the programmer desires to separate the logical portions of his program. The segment number, s, is two digits. If the program is all in one segment, the segment number is 01. The START instruction serves the same purpose for the first segment of a program as the SEGMENT START does for all the others.

The tag of the starting location of the program must be specified in absolute. When the assembled program is read into memory, the Sequence Register is set at this starting location, which is printed on the Console Typewriter as

BEG ----.

The machine will stop after this printout if Console Breakpoint Switch 1 is set (see Figure 4); otherwise control is immediately transferred to the starting location of the program.

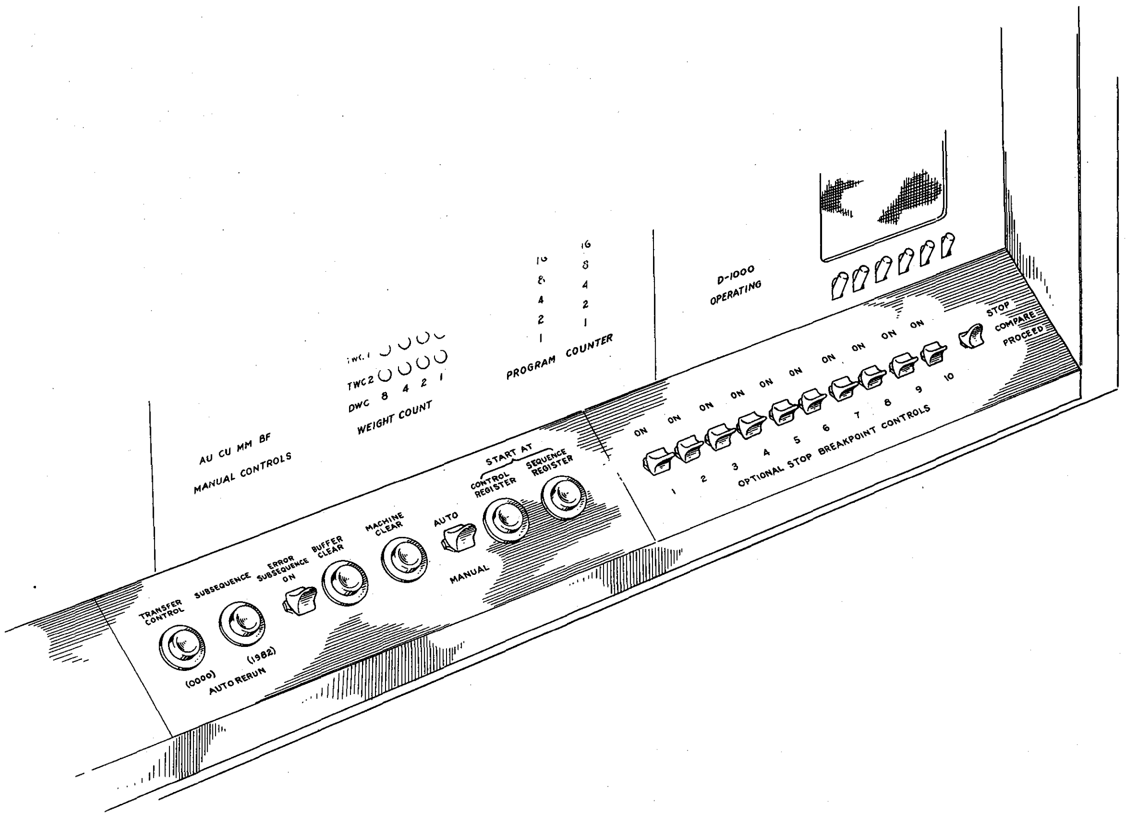


Figure 4. Central Console Switch Panel

SEGMENT START/s □

s - Segment Number

When more than one segment is used, the coding for each segment is immediately preceded by a SEGMENT START instruction. This sets up the Beginning of Segment Identification block on the program tape (Figure 14). This instruction is used for all segments of a program except the first, where the START instruction serves the same purpose. As noted, the segment number has two digits. Segments are written on the Program Tape in the order they are received by the Assembly Program. No attempt is made to arrange segments sequentially if they are not received that way.

A stem, defined once in a program, maintains its value through all the segments of a program unless a further TAG instruction reassigns the value. Hence, simple communication is possible between the segments of a program.

READ DIRECTED SEGMENT: RDS/ $\left\{\begin{matrix} B \\ F \end{matrix}\right\}$ /T/s/Y □

The RDS instruction is an optional instruction. If written in the program at the point where another segment is needed, the Assembly Program will automatically insert a routine during assembly which will search for and read into memory the segment indicated and transfer control to the address indicated. This instruction has an identification tag and the next 35 locations in memory must be left empty. The routine generated is placed in these locations by the Assembly Program and these locations must be avoided by the programmer desiring to make use of the RDS instruction. Locations 1972-1977 are used during execution of the RDS instruction and, therefore, cannot contain information to be preserved from one segment to another.

The first zone indicates whether the segment needed may be found by reading backward (B) or forward (F) on the Program Tape. If, for some reason, this is not known, the inserted routine will search forward, then backward if necessary. The programmer, however, generally knows the direction of search. Tape number is indicated by two digits in zone 2 of the RDS instruction. This is the tape drive of the Program Tape when the assembled program is run. Note that if the Assembly Program is used to operate the assembled program immediately after assembly, the Magnetic File Unit is 03. Zone 3 contains two digits to indicate the segment desired. If either zone 2 or 3 is missing, the words

ILLEGAL
RDS ← Read directed segment

are printed on the Console Typewriter and the instruction is not processed. The last zone contains the starting location after the segment has been read into memory.

When a segment has been read into memory by the assembled program, this location is printed on the Console Typewriter as

BEG ----.

The machine will stop after printing this out if Console Breakpoint Switch 1 is set (Figure 4). If no starting location is given, the words

NO BEGIN

are printed on the Console Typewriter, the machine stops unconditionally and the programmer is responsible for restarting his program. If, for some reason, the segment cannot be found,

NO SEG --

is printed and the machine stops. If this RDS instruction is not used in a program having more than one segment, the programmer assumes the responsibility for searching for and reading into memory his own segments as needed.

Once the proper segment has been located on the program tape, it is only necessary to read in a word and subsequence to it. This word causes the read-in of the words following it which are in consecutive locations. At the end of each block, there is a sentinel subsequence call to 1984 or 1985. Hence, these locations must be loaded with appropriate instructions. For the first segment written on tape, these locations contain a Sequence Change instruction to 1972. A Read instruction with the proper tape number must, therefore, be stored in 1972 for automatic read-in. The word stored on tape by the Assembly Program at the end of each segment is a subsequence call to 1980. This word is read into memory and performed when the rest of the segment is in memory. Proper sequencing of instructions may be effected by loading 1980 with an instruction which transfers control to the location of the first instruction to be performed after the segment is in memory.

As an example, the coding produced by the Assembly Program to load a program reading forward from tape 3 which starts at

0011 is:

1972	RFD/03 □
1973	TIS/1975/1/1975 □
1974	SCS//1973 □
1975	□ (Place for control word)
1976	A/BEG(sp)0011 □
1977	OST//1 □
1980	PRA/1976/0011/1977 □
1984	SCS//1972 □
1985	SCS//1972 □

HOLLERITH CARD FORMAT

A sample card form is shown in Figure 5. Both instructions and constants use the same card format. A card has six fixed fields and a variable field. The six fixed fields are:

(1) Program Identification - columns 1-5. The program identification is a 5-alphanumeric-character field for identifying the cards of a program.

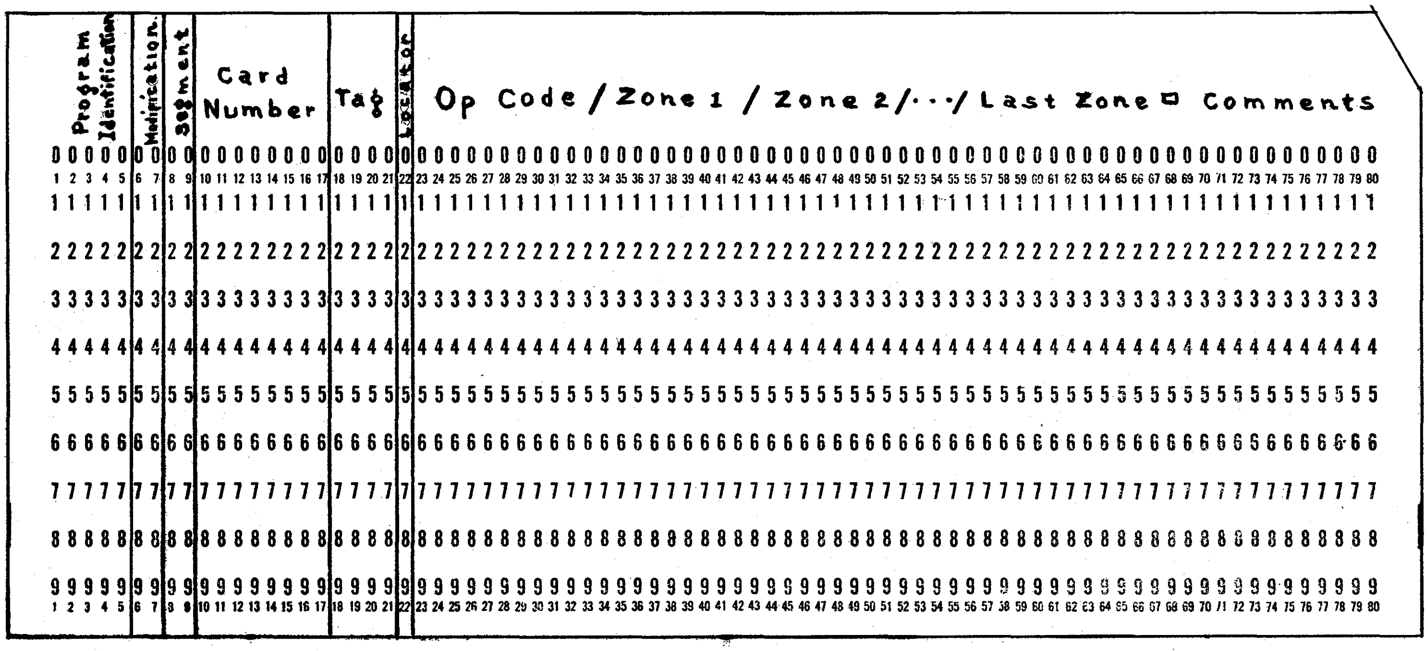


Figure 5. Assembly Program Input Card

(2) Modification Code - columns 6 - 7. The modification number on all cards of a program is set initially. Every time a program is modified by adding a set of correction cards at the end or by replacing a card, the modification number on these cards is changed. Any alphanumeric characters may be used for the purpose.

(3) Segment Identification - columns 8-9. The segment identification consists of two decimal digits.

(4) Card Number - columns 10-17. The card number is a serial number identifying each card. Columns 10-14 are used for consecutive numbering and columns 15-17 contain fractional numbering which permits the insertion of words into the program after all of the cards have been prepared. As an example, if an insertion of two cards is put between cards numbered 15,427 and 15,428, the four cards might have the following numbers in columns 10-17.

Column	10	11	12	13	14	15	16	17
First Card	1	5	4	2	7	0	0	0
Insert A	1	5	4	2	7	1	0	0
Insert B	1	5	4	2	7	2	0	0
Second Card	1	5	4	2	8	0	0	0

A further insertion at this point will result in

Column	10	11	12	13	14	15	16	17
First Card	1	5	4	2	7	0	0	0
Insert A	1	5	4	2	7	1	0	0
Insert A'	1	5	4	2	7	1	1	0
Insert B	1	5	4	2	7	2	0	0
Second Card	1	5	4	2	8	0	0	0

Figure 6

(5) Identification Tag - columns 18-21. The identification tag is the absolute or relative address of the instruction or constant on the card. If consecutive cards contain words for consecutive locations, loading time for the assembled program is shortened considerably.

(6) Locator - column 22. The locator column, L, contains an S if the instruction on the card is to be a sentinel. Otherwise, nothing appears in this column.

Starting at column 23, the instruction or constant is written, punctuated by slashes, and ended by a square. Comments may follow. Cards which contain only comments have a card number but no tag, and a square appears in column 23. These cards are not processed, nor are any comments written on tape.

INPUT CONVERTER OPERATION

The program cards of several different programs to be assembled may be fed into the Input Converter as one large deck. The program identification in card columns 1 - 5 is used to distinguish among programs on the magnetic tape produced by the Input Converter. Care must be taken to insure that the same program identification appears on every card of a program and that each program assembled contains a unique program identification.

A special deck of five cards is used to indicate the end of the last program to be assembled on the Input Converter tape. These five cards must have the alphanumeric characters E O F R I in columns 1 - 5. The information punched on the rest of the card is immaterial. These cards always follow the last card of the last program to be assembled.

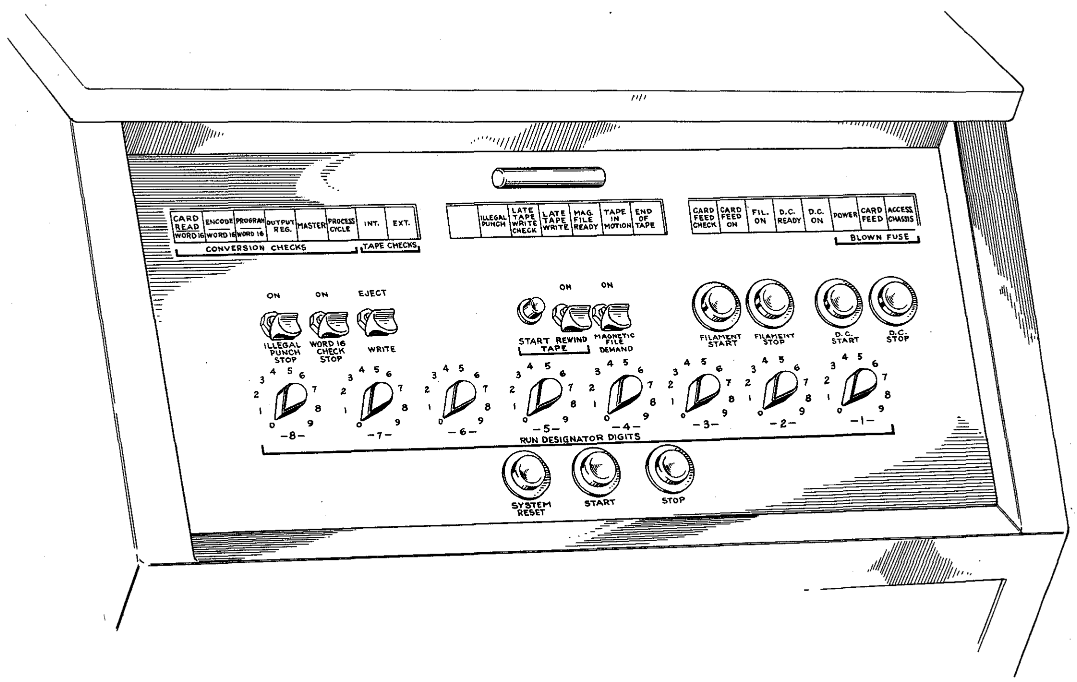


Figure 7. Input Converter Console

The Word 16 Check switch is set to stop for a converter error. The Eject switch is set to eject a card with an error. If a converter error occurs, the deck of cards in the input hopper of the Input Converter is moved back and the card with the converter error and those ejected with it are placed in front of the deck to be processed. Then the Converter Start button is pressed and processing continues.

The Input Converter reads all information except the card number in 6-bit code from the Hollerith card. The Assembly Program will perform any necessary editing. The two Input Converter control panels are wired as shown in Figures 8 and 9 for use with the Assembly Program.

DATAmatic

Layout Form — Input Converter, Model 1200

(CARD READING CONTROL PANEL)

Title ....

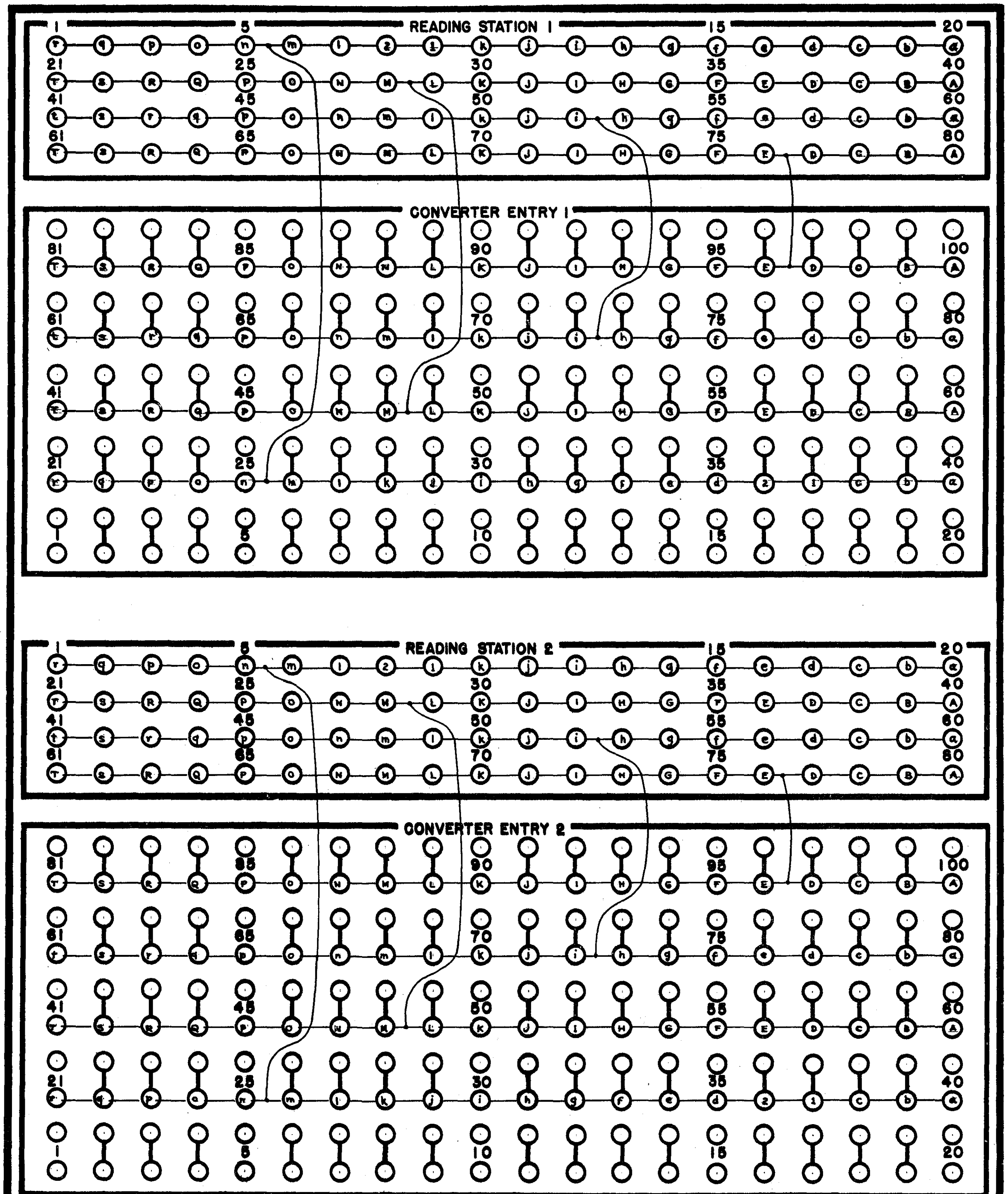
Prepared by .... For Program Assembly Program

By Programmer .... Checked by ....

Date .... Remarks ....

....

Modification .... Page ..... of .....



1056-657

FIGURE 8

PRINTED IN U.S.A.

DATAmatic **Layout Form — Input Converter, Model 1200**

(CONVERSION CONTROL PANEL)

Title..

Prepared by...For Program...Assembly Program

By Programmer...Checked by.....................................

Date..Remarks...

....

Modification..Page.................of...........

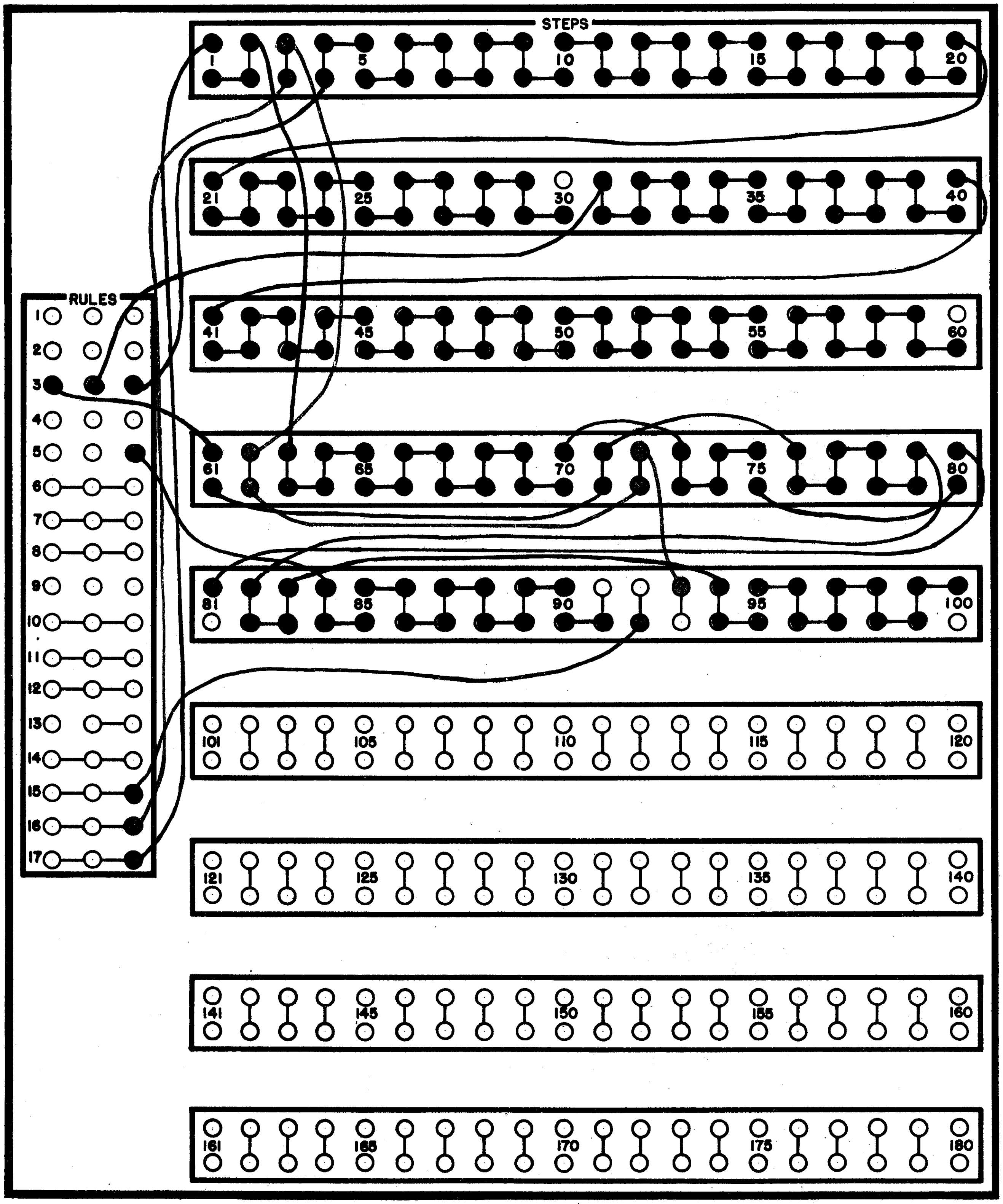


Figure 9

ASSEMBLY PROGRAM OPERATION

The operating procedure for the Assembly Program is outlined in Figure 11. The input to the Assembly Program is the magnetic tape produced by the Input Converter. The output from the Assembly Program is a Program Tape which contains the operating form of the assembled program and a Console Typewriter listing of any errors detected during assembly. The Input Tape must be mounted on the Magnetic File Unit which is selected internally by the code 01, and the Program Tape must be mounted on the Magnetic File Unit which is selected internally by the code 03. The tapes are positioned to the beginning by the controls on the Magnetic File Units. Both tapes may have other information on them. On the input tape, the Assembly Program searches for the START instruction containing the Program Search Code read into 1978. On the Program Tape, the Assembly Program searches for the end-of-reserved-information block. This presupposes that if no other program or reserved information is on the Program Tape, the first block on the tape is an end-of-reserved-information block. Such a block has the following information in words 1-6:

```
1    E 0 0   0 0 0   0 0 0   0 0 0
2    G G G   G G G   G G G   G G G
3    0 0 0   0 0 0   0 0 0   0 0 0
4    0 0 0   0 0 0   0 0 0   0 0 0
5    0 0 0   0 0 0   0 0 0   0 0 0
6    Z   Z   Z   Z   Z   Z   Z   Z
```

End of reserved info. block.

Figure 10

Operating Procedure for Assembly Program

Ⓐ

↓

Prepare Program To Be Assembled

Write program using available instructions.
Punch program on Hollerith Cards, one instruction per card.
Put program cards on Input Converter, using Assembly Program control panel.

↓

Load Assembly Program in Memory

Mode of Operation With Assembly Program

From Magnetic Tape	From Hollerith Cards	From Paper Tape
1) Load paper tape with correct routine to find Assembly Program via Console Typewriter.	1) Process cards with Assembly Program in Input Converter (use special control panel).	Load paper tape into memory starting at 0000.
2) Type ASSEMBLY in 19~~78~~63 via Console Typewriter.	2) Load paper tape with correct Assembly Loading Routine in 1400.	
3) Transfer control to paper tape.	3) Type ASSEMBLY in 1978 via Console Typewriter.	
	4) Type SCS//1400 □ [B 00 000 400 000] in location 0000.	
	5) Press Start at 0000 button on Central Console.	

↓

Load Tapes

Input Converter Tape with program to be assembled on Unit 01.
Program Tape for Assembled Program on Unit 03.
Rewind both tapes.

↓

Set Central Console Switches

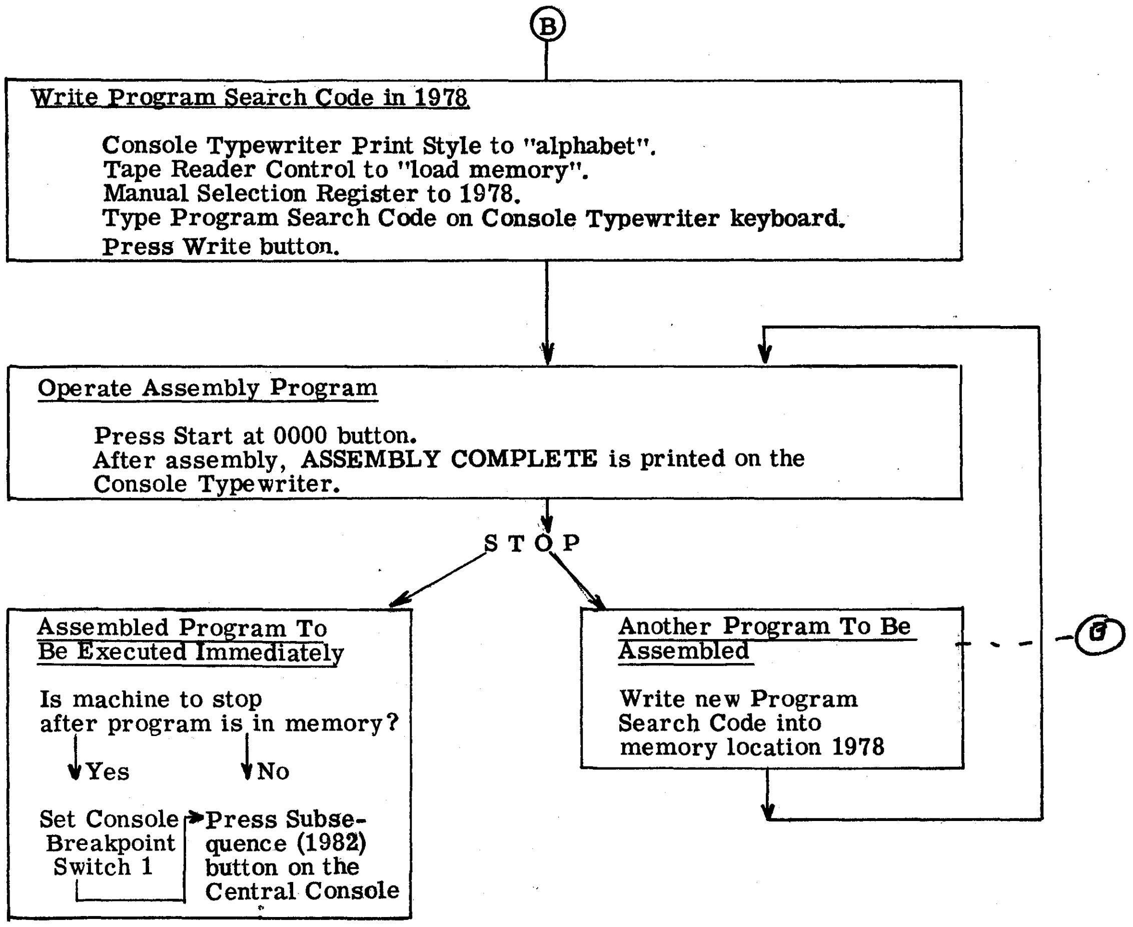
Console Typewriter Space Suppress to "on".
Console Typewriter Special Symbols to "off".
Console Typewriter Auto Instruct Print to "off".
Console Error Subsequence to "on".

↓

Ⓑ

Figure 11

Page 1



Write Program Search Code in 1978

Console Typewriter Print Style to "alphabet".
Tape Reader Control to "load memory".
Manual Selection Register to 1978.
Type Program Search Code on Console Typewriter keyboard.
Press Write button.

Operate Assembly Program

Press Start at 0000 button.
After assembly, ASSEMBLY COMPLETE is printed on the Console Typewriter.

Assembled Program To Be Executed Immediately

Is machine to stop after program is in memory?

Set Console Breakpoint Switch 1

Press Subsequence (1982) button on the Central Console

Another Program To Be Assembled

Write new Program Search Code into memory location 1978

Figure 11
Page 2

Loading the Assembly Program

The Assembly Program is contained in one segment. It can be loaded into the High-Speed Memory in one of three ways: [illegible]

(1) Assembly Program on magnetic tape, start at 1977 Assembly Read-in Routine on short paper tape. The Assembly Program is kept on magnetic tape in the same way as any assembled program, i.e., with self-loading instructions. A paper tape finds the Assembly Program on tape and initiates its read-in. To do this, the word ASSEMBLY is typed into 19~~78~~63 on the Console Typewriter and the paper tape is read into memory. Control is then transferred to the paper tape routine. [illegible]

(2) Assembly Program on a deck of cards, Assembly Loading Routine on short paper tape. The deck of cards on which the Assembly Program is punched is put through the Input Converter onto magnetic tape. The Assembly Loading Routine is read into the machine via paper tape starting at location 1400 and an SCS//1400□ instruction is read into 0000. Press the Start at 0000 button and the loading of the Assembly Program into memory is done by the program on the paper tape.

(3) Assembly Program on paper tape. The Assembly Program is read into the machine on paper tape, starting at location 0000. This much slower procedure does not require the use of an Input Converter Tape or a separate loading routine.

Starting the Assembly Process

After the Assembly Program is in memory and the program to be assembled is on the Input Converter Tape, the identification of the program to be assembled is typed into memory using the Console Typewriter. This is accomplished by setting the Manual Register

Selector to 1978 (the location of the program identification in memory) and the Print Style switch to Alphabet. Type the Program Search Code and enough & O's zeros to form an 8-character word and depress the Write button. The Console Typewriter switch panel is illustrated in Figure 12.

what about load memory button ?

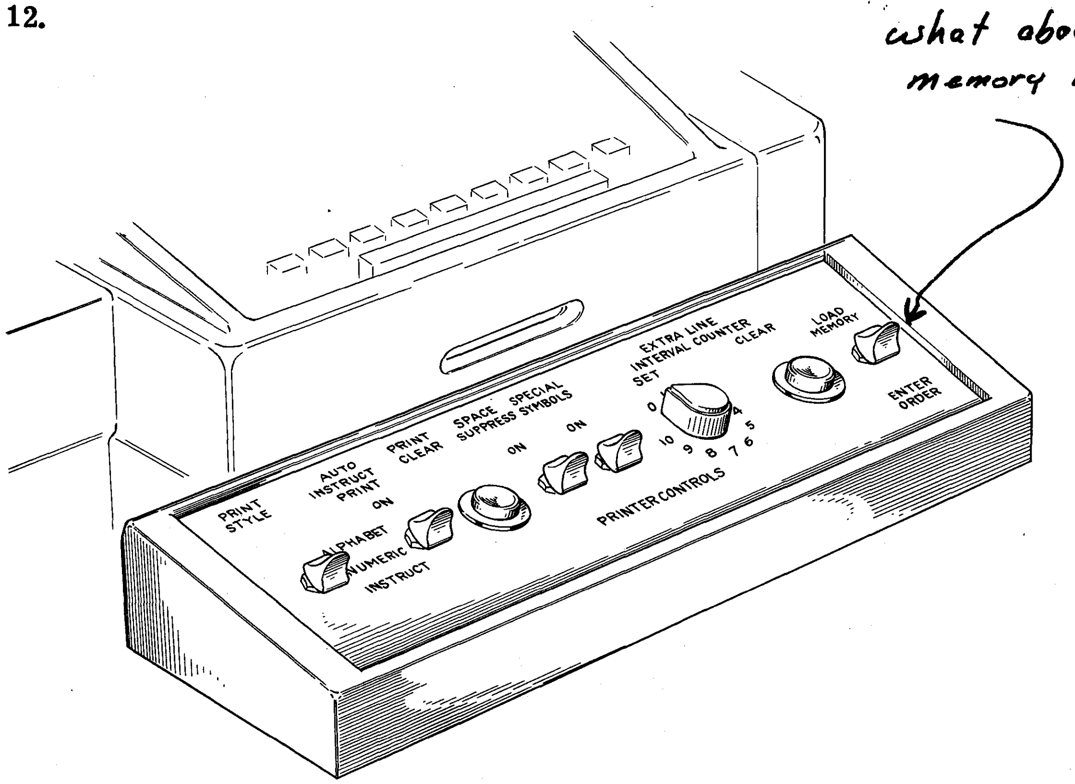


Figure 12. Console Typewriter Switch Panel

<u>The Assembly Program is started by pressing the Start at 0000 button</u>. The Assembly Program locates the program to be assembled on the Input Tape by searching for the START card containing the Program Search Code written into 1978. If no program can be found, the words

PROGRAM
MISSING

are written on the Console Typewriter. The place for the program on the Program Tape is found by finding the "end-of-reserved-information" block on the Program Tape. <u>The end-of-reserved-information block is erased and the assembly process is started.</u>

Programming Error Provisions

At the start of the assembly of each program or segment of a program, the identification of the program and segment number are printed on the Console Typewriter. When an error is detected, the Assembly Program lists the card number, the identification tag of the word (the contents of the tag field), and the error type on the Console Typewriter. The types of errors detected by the Assembly Program and the corresponding printouts are summarized in Table IV (pages 30-31). A sample error printout format appears in Figure 13. In order to obtain these error printouts in readable form, the Auto Instruct Print and Special Symbols switches must be in the OFF position and the Space Suppress switch must be in the ON position on the Console Typewriter (see Figure 12). The contents of the card with the error are disregarded by the Assembly Program; that is, there is no corresponding word on the Program Tape. The detection of an error does not necessarily stop the assembly process.

If a START card is found which has the correct Program Search Code in the first zone and then a square (no segment number given), the words

```
NO SEGMT
NUMBER
```

are printed on the Console Typewriter and the machine stops. This same printout is also issued if the segment number is not given in the SEGMENT START instruction. The Assembly Program cannot proceed after this type of error. If the START instruction contains more than three zones, i.e., there is no square after the starting location, the words

```
ILLEGAL
START
```

are printed on the Console Typewriter and the machine stops. At this point, the Program Search Code, the identification and modification numbers from columns 1-7 on the card, and the segment number have been printed. If it is known that the only error was a slash punched in place of a square at the end of the third zone, press the Sequence Register button on the Central Console. This will cause a printout of the starting location of the program as indicated in the START instruction,

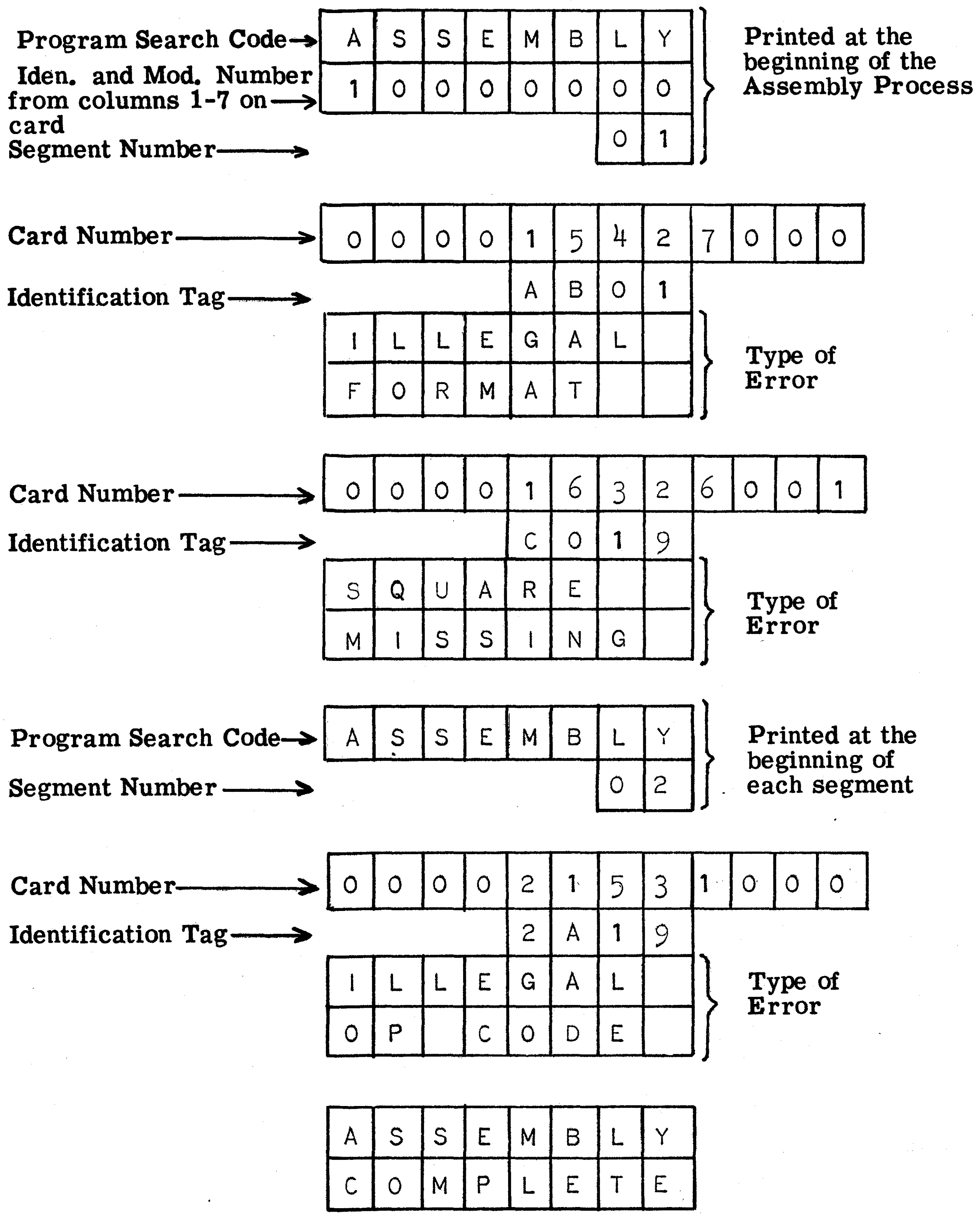


Figure 13. Sample Error Printout Format

PRINTOUTS DURING THE ASSEMBLY PROCESS

Printout	Reason for Printout	Instruction for Programmer
PROGRAM MISSING	1) The program identified by the program search code written into 1978 cannot be found on the input tape on Magnetic File Unit 1. 2) The assembled program cannot be found on the program tape.	Machine stops unconditionally. Check program search code. Check for proper tape number.
ILLEGAL START	No square appears after the last zone of the START instruction.	If segment number printed on preceding line is correct, press Control Register button.
	Starting location as given is printed to enable programmer to determine if this is the correct START card.	If starting location is correct, press Control Register button; if not, Assembly Program cannot continue.
NO SEGMT NUMBER	The segment number in the SEGMENT START instruction is not given.	The Assembly Program cannot continue after this type of error.
ILLEGAL RDS	Either the Magnetic File Unit or the segment number are not given in the RDS instruction.	The RDS instruction is ignored, and the routine to search for and read into memory the segment wanted is not in the assembled program.
TAG ILLEGAL	1) Stem was not defined previous to use in the program. 2) The value assigned to a stem plus the digits which follow it add up to a tag greater than 1999.	The contents of the card specified by the card number and identification tag preceding each of these errors is ignored. No word corresponding to the card appears on magnetic tape. The programmer is responsible for correcting the incorrect card by re-assembly or by making use of an appropriate service routine.
ILLEGAL FORMAT	The form of the instruction or constant does not comply with its specifications.	
TOO MANY ZONES	More zones appear on the card than are specified by the instruction. This type of error may occur if a square did not terminate the end of the instruction, but did appear elsewhere on the card.	
ILLEGAL OP CODE	The operation code is not one listed as applicable to the Assembly Program.	
SQUARE MISSING	A square has not been used to terminate an instruction or constant to be assembled. Note the special printout (ILLEGAL START) when this occurs with the START instruction.	
ILLEGAL CHARACTR	The type of character which should appear in a zone is not there.	
TOO MANY CHARACTS	There are more characters in a zone than specified. This is particularly significant with constants.	

TABLE IV
Page 1

Printout	Reason for Printout	Instructions for Programmer
ASSEMBLY COMPLETE	When a program is assembled and on magnetic tape ready for use, this is printed on the Console Typewriter.	If the program is to be used immediately, press Subsequence (1982) button. a) Program to start immediately after loading into memory, do not set Breakpoint Switch 1. b) Program to stop after it is in memory, set Breakpoint Switch 1.
RERUN OK	A transfer weight count error on reading information into the buffer from the Input Converter tape was encountered. Rerun brought the correct information into the memory.	This printout is for statistical purposes only and has no bearing on the operation of the program.
TWC ERR	A transfer weight count error in reading information into the buffer from the Input Converter tape was encountered. Rerun could not bring in the correct information.	The card number and identification tag are printed preceding this error printout. It is possible that one or the other is not correct. The information from this block is not processed by the Assembly Program but operation continues.

PRINTOUTS DURING OPERATION OF THE ASSEMBLED PROGRAM

BEG----	1) This indicates the starting location of the first segment read into memory. 2) The same printout will appear for each segment read into memory by an RDS instruction.	If Breakpoint Switch 1 has been set and the machine stops, press the Control Register button on the Console to start program operation. This is automatic if Breakpoint Switch 1 is not set.
NO BEGIN	1) The starting location was not indicated in the START instruction. 2) The starting location was not indicated in the RDS instruction.	The machine stops regardless of breakpoint switch settings. The programmer is responsible for starting his program at the proper location.
NO SEG--	The segment called for cannot be found on the tape indicated.	Machine stops unconditionally. Check segment identification. Check for proper tape number.

TABLE IV
Page 2

after which the machine will stop again. If this printout is correct, the Assembly Program continues to process the program when the Control Register button on the Central Console is pressed .

During the assembly process, the following types of errors are detected (see Table IV):

(1)	TOO MANY ZONES	
(2)	SQUARE MISSING	(Cannot determine the end of significant information on card)
(3)	ILLEGAL CHARACTR	
(4)	TOO MANY CHARACTS	
(5)	ILLEGAL OP CODE	(Not Rank 0 or Assembly Control instruction)
(6)	TAG ILLEGAL	(Tag greater than 1999 or no assignment to letters previous to use)
(7)	ILLEGAL FORMAT	

To correct errors, correction cards may be placed at the end of the program deck, or the faulty card may be corrected. The program is then reassembled to correct the errors. An alternate procedure for program correction is to make corrections on the Program Tape using a service routine. For this latter method, any changes and/or insertions to be made follow the procedure for the use of the routine chosen. When the assembly is complete, the words

ASSEMBLY
COMPLETE

are printed on the Console Typewriter.

Resetting the Assembly Program

The Assembly Program may be used to process more than one program at a time. When the words

ASSEMBLY
COMPLETE

are printed, the new Program Search Code is written into 1978 on the Console Typewriter and the Start at 0000 button on the Central Console is pressed.

Program Tape

Programs prepared by the Assembly Program are placed on the Program Tape, i. e., the tape on Magnetic File Unit 03. Each program is identified by its unique Program Search Code (the contents of the first zone of the START instruction). No attempt is made to order the programs by their search codes. A program on the Program Tape has the following logical sections:

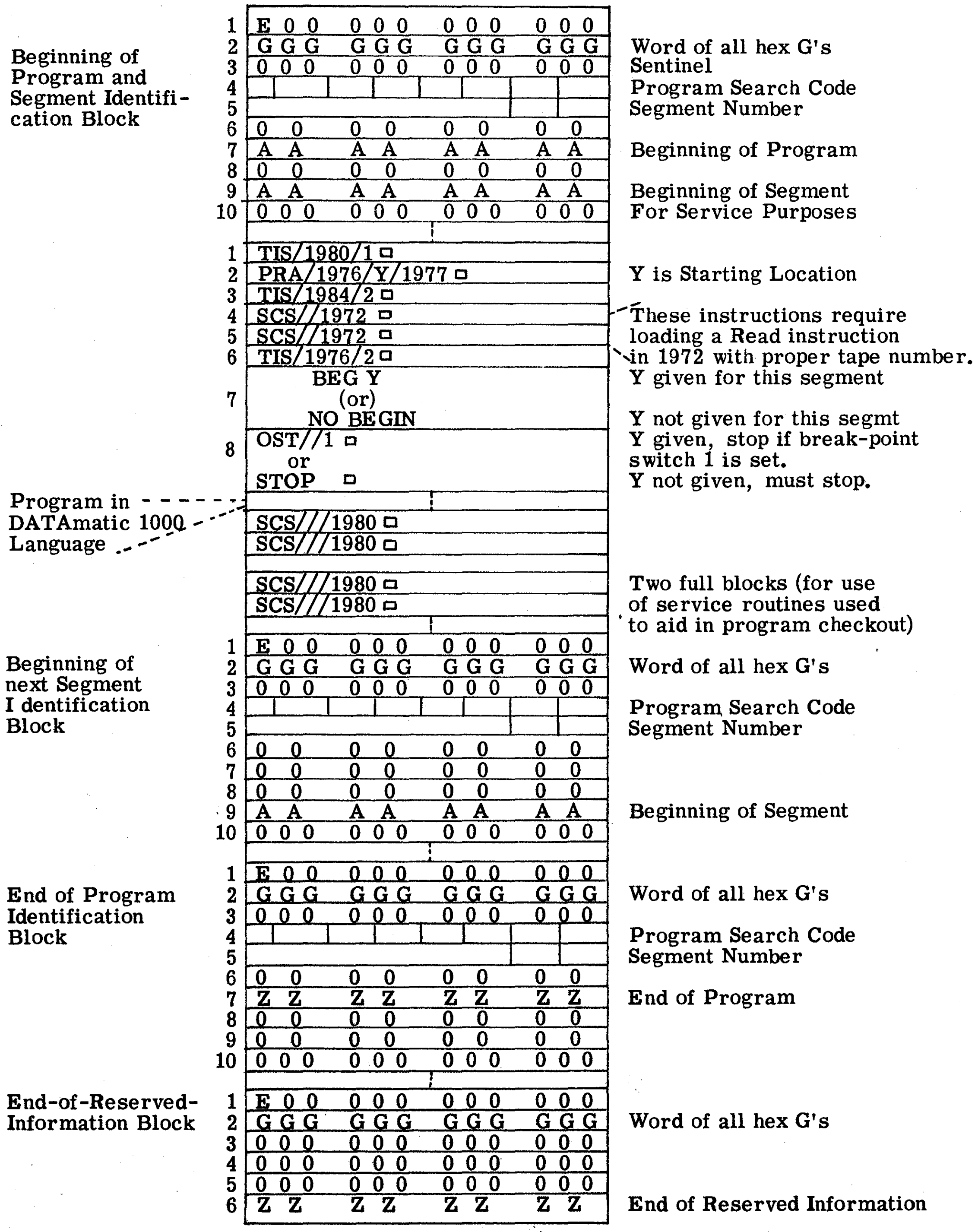
(1) Program and Segment Identification block
(2) Instructions for significant locations for automatic read-in of first segment
(3) Program in DATAmatic 1000 language with control words for read-in
(4) Two blocks of SCS///1980□ to allow program modification without reassembly
(5) Next Beginning of Segment Identification block.

Sections 5, 3, and 4 are repeated until the end of the program is reached, at which time the final block is an end-of-program block. Following this is an end-of-reserved-information block. The format is illustrated in Figure 14.

OPERATING PROCEDURE FOR THE ASSEMBLED PROGRAM

If it is desired to read the program into memory immediately, type the Program Search Code in 1978 and press the Subsequence (1982) button. This will cause a search of the Program Tape for the assembled program. If the program cannot be found, the words

PROGRAM
MISSING



Beginning of Program and Segment Identification Block	1	E 0 0 0 0 0 0 0 0 0 0 0	
	2	G G G G G G G G G G G G	Word of all hex G's
	3	0 0 0 0 0 0 0 0 0 0 0 0	Sentinel
	4		Program Search Code
	5		Segment Number
	6	0 0 0 0 0 0 0 0	
	7	A A A A A A A A	Beginning of Program
	8	0 0 0 0 0 0 0 0	
	9	A A A A A A A A	Beginning of Segment
	10	0 0 0 0 0 0 0 0 0 0 0 0	For Service Purposes
	1	TIS/1980/1 ◻	
	2	PRA/1976/Y/1977 ◻	Y is Starting Location
	3	TIS/1984/2 ◻	
	4	SCS//1972 ◻	These instructions require loading a Read instruction in 1972 with proper tape number.
	5	SCS//1972 ◻	
	6	TIS/1976/2 ◻	
	7	BEG Y (or) NO BEGIN	Y given for this segment Y not given for this segmt
	8	OST//1 ◻ or STOP ◻	Y given, stop if break-point switch 1 is set. Y not given, must stop.
Program in DATAmatic 1000 Language			
		SCS///1980 ◻	
		SCS///1980 ◻	
		SCS///1980 ◻	Two full blocks (for use of service routines used to aid in program checkout)
		SCS///1980 ◻	
Beginning of next Segment Identification Block	1	E 0 0 0 0 0 0 0 0 0 0 0	
	2	G G G G G G G G G G G G	Word of all hex G's
	3	0 0 0 0 0 0 0 0 0 0 0 0	
	4		Program Search Code
	5		Segment Number
	6	0 0 0 0 0 0 0 0	
	7	0 0 0 0 0 0 0 0	
	8	0 0 0 0 0 0 0 0	
	9	A A A A A A A A	Beginning of Segment
	10	0 0 0 0 0 0 0 0 0 0 0 0	
End of Program Identification Block	1	E 0 0 0 0 0 0 0 0 0 0 0	
	2	G G G G G G G G G G G G	Word of all hex G's
	3	0 0 0 0 0 0 0 0 0 0 0 0	
	4		Program Search Code
	5		Segment Number
	6	0 0 0 0 0 0 0 0	
	7	Z Z Z Z Z Z Z Z	End of Program
	8	0 0 0 0 0 0 0 0	
	9	0 0 0 0 0 0 0 0	
	10	0 0 0 0 0 0 0 0 0 0 0 0	
End-of-Reserved-Information Block	1	E 0 0 0 0 0 0 0 0 0 0 0	
	2	G G G G G G G G G G G G	Word of all hex G's
	3	0 0 0 0 0 0 0 0 0 0 0 0	
	4	0 0 0 0 0 0 0 0 0 0 0 0	
	5	0 0 0 0 0 0 0 0 0 0 0 0	
	6	Z Z Z Z Z Z Z Z	End of Reserved Information

Figure 14. Program Tape Format

are printed and the machine stops. When the correct program is found, its Program Search Code is printed on the Console Typewriter and the program (or first segment) is read into memory. After the program is in the High-Speed Memory, if the starting location, Y, was given, the words

BEGinY

are printed. If the programmer wishes the machine to stop for any reason after the program is loaded and prior to its execution, con-sole Breakpoint switch 1 is set. If Breakpoint switch 1 is not set, control is transferred to the program in memory without stopping. If the starting location, Y, was not given, the words

NO BEGIN

are printed and the programmer is responsible for starting his program.

When the assembled program is not to be used immediately, the Program Tape may be mounted on any Magnetic File Unit and the procedure for use of the standard input program used. If the RDS instruction is not used, the programmer must load in 1980 a Sequence Change instruction to the starting location to be used after each segment is read into memory. This is done automatically for the first segment on the tape.

Appendix A

Hollerith Card Format

It may be desirable, at some installations, to use a fixed-field card format which will fit most instructions. The Hollerith Card will then have the following format (see Figure 5A).

1) Program Identification -- columns 1 - 5
2) Modification Code -- columns 6 - 7
3) Segment Identification -- columns 8-9
4) Card Number -- columns 10-17
5) Tag -- columns 18-21
6) Locator -- column 22
7) Operation code -- columns 23-25
   Slash -- column 26
8) Zone 1 -- columns 27-30
   Slash -- column 31
9) Zone 2 -- columns 32-35
   Slash -- column 36
10) Zone 3 -- columns 37-40
   Square (□) -- column 41

Columns 42-80 may be used as the programmer desires.

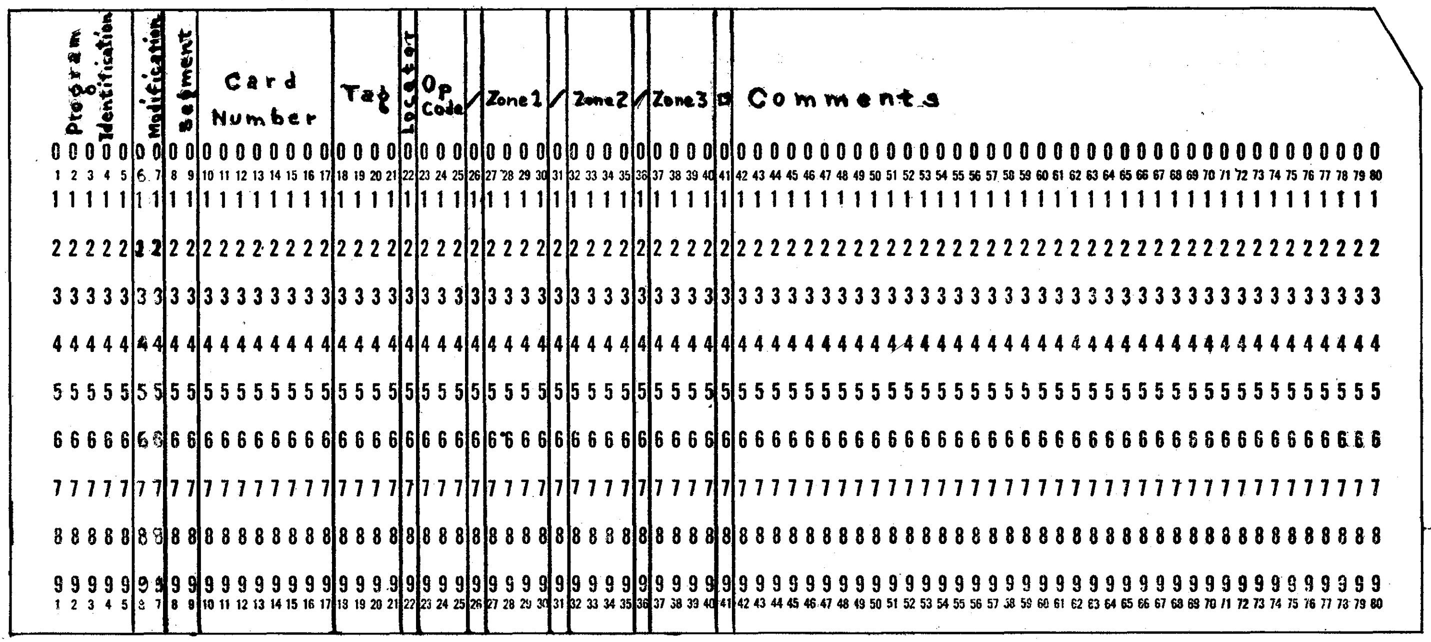


Figure 5A. Assembly Program Input Card - Fixed Fields

Programmer's Language

The fixed-field format introduces the following changes and notations in Programmer's Language.

Instructions which have more than three zones, that is, Transfer and Select (TSA, TSB, TSD, TSS), Double Transfer and Select (DTA, DTB, DTD, DTS) and the corresponding Bypass Interlock orders cannot be written in the fixed-field format. All constants, Alphabetic, Signed Numeric, and Unsigned Numeric, cannot be written in the fixed-field format. The Assembly Control Instructions START, SEGMENT START, and RDS, also require special cards. The instructions discussed in this paragraph are written in a varied-length field from column 23 on.

The second zone of the Shift Instructions must be modified so that it always contains four alphanumeric characters. This is done by placing enough zeros to the left of the number to make a total of four characters. Hence a shift of four (4) alphabetic places would be written 004A; ten (10) numeric places will be written 010N.

A zone must always contain four alphanumeric characters. Hence, in both absolute and relative coding, all four digits of a tag must be written. The major difference between variable and fixed-field format is that in the fixed-field format the TAG instruction must specify four characters in the first zone. Since there are only two zones in the TAG instructions, the third-zone field contains four zeros or four blanks. An example of coding is:

```
                      TAG/1A00/0500/0000 □
                      TAG/2A00/0560/0000 □
                      TAG/CD00/0450/0000 □
CD25                  ADD/1A00/2A05/1A10 □
CD26                  BAR/2A30/2A10/0000 □
CD27                  SLW/2A05/003N/2A05 □
CD28                  SCS/0000/CD49/0000 □
```

This is equivalent to the absolute coding:

```
0475          ADD/0500/0565/0510 □
0476          BAR/0590/0570/0000 □
0477          SLW/0565/003N/0565□
0478          SCS/0000/0499/0000 □
```

Note that <u>only</u> if four characters are used in the first zone of the TAG instruction, does the Assembly Program consider that fixed fields are used.

All other specifications and results of the Assembly Program remain unchanged.