RARE MANUAL

THE DESCRIPTOR

a definition of the B 5000 information processing system

BATSON

DO NOT STEAL

BULLETIN 5000-20002-P
FEBRUARY 1961

Gus Bailey

THE DESCRIPTOR

– a definition of the B 5000 Information Processing System

Sales Technical Services
Equipment and Systems Marketing Division

Burroughs Corporation
Detroit 32, Michigan

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS (continued)

APPENDIXES

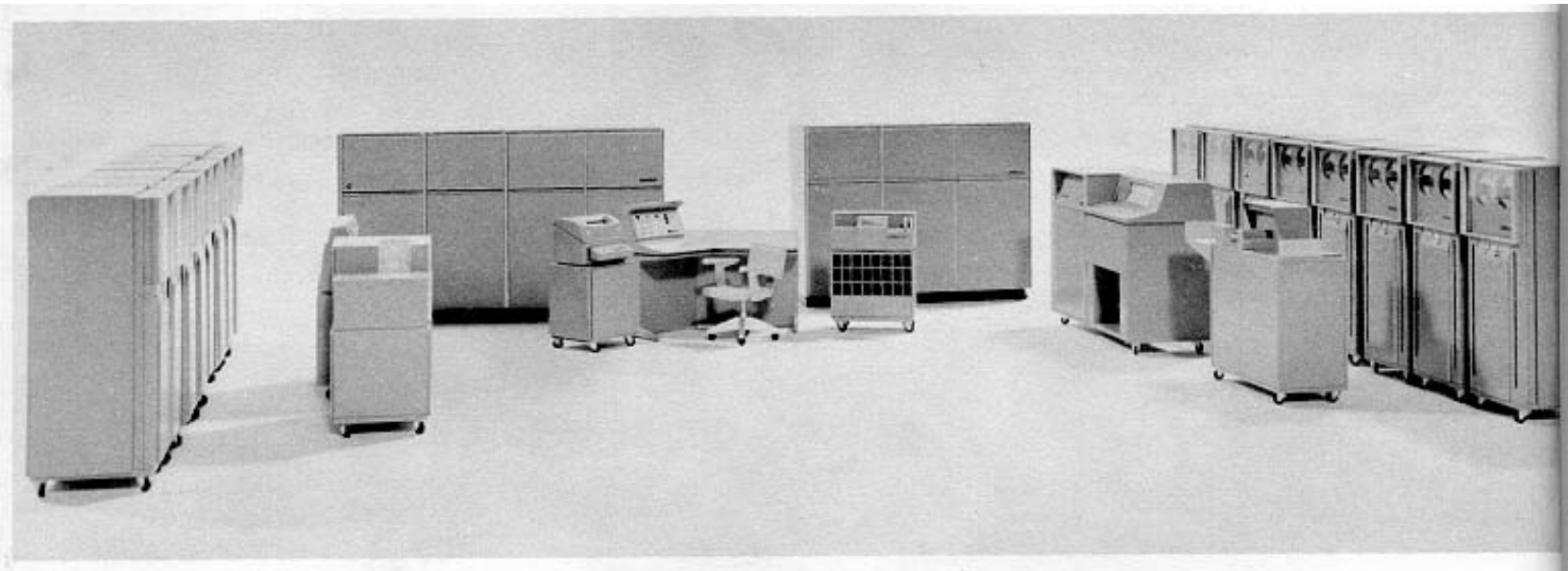
APPENDIXES (continued)

APPENDIXES (continued)

LIST OF ILLUSTRATIONS

FIGURES

TABLES



The B 5000 Information Processing System

SECTION 1

INTRODUCTION

As the title implies, this manual contains a description in some detail of the B 5000 System and its operation. It is intended to provide technical information for those directly concerned with data processing—managers of data processing systems, programmers, and management personnel who are acquainted with the problems and concepts of electronic data processing. It is not intended as a teaching manual or programming primer. Without delving into such elements of computer design as circuitry and machine logic, this handbook explains how the B 5000 achieves its remarkable flexibility and efficiency through a comprehensive systems approach to problem solving.

The fact that the B 5000 works as a system—rather than only an advanced set of hardware—has determined the organization of this manual.

The source languages, ALGOL and COBOL, are presented first because they are the link between the user and the system, the statement of his problem and its ultimate solution. The next section, System Characteristics, discusses the way the B 5000 functions as a system and processes a program. Over-all coordination and control of processing, so important to total production through maximum use of the B 5000 components, is supplied by the Master Control Program, which is described next. Finally, the operational characteristics and features of the components which implement this unique system approach are specified in detail. Two appendixes provide comprehensive definitions of the programming languages, ALGOL and COBOL, including examples of their use. A glossary is included for reference to definitions of terms used with the B 5000 System.

The Descriptor is one of a series of technical publications which will give the user complete information on the equipment and its use. The B 5000 Concept Manual introduces the reader to the new design approach that integrates hardware and software—automatic programming and advanced operating techniques—into a unified package. This first edition of the Descriptor will be augmented by the publication of programming primers, reference manuals, and advanced programming manuals for ALGOL and COBOL.

The B 5000 was designed as a complete system, *combining* components and built-in programming aids, to bring the user simplified programming, ease of operation, and complete freedom of system expansion.

The programmer works in the language of his problem, either algebraic notation or English narrative statements. The B 5000 has compiler-oriented machine language and logic which accept the common languages of ALGOL and COBOL. Because the machine language of the B 5000 is similar to these problem languages, compilation time is reduced and object-program redundances eliminated. The Processor operates on either single characters or complete words, alphanumeric or binary information, and the programs it uses are extremely compact. Since the B 5000 handles memory and input/output unit assignments, segmentation of programs, and subroutine linkages automatically, the programmer is freed of many arduous tasks and the likelihood of error is reduced. Correction of programs is done at the source-language level and is further simplified by integral debugging aids.

Operator intervention is nearly eliminated because complete management of the system is assumed by the Master Control Program, a comprehensive operating system that is recorded on the Storage Drum of the B 5000 System and provides simultaneous input/output operations and Multi-Processing. By controlling the sequence of processing, initiating all input/output operations, and providing automatic handling procedures to meet virtually all processing conditions, the Master Control Program can obtain maximum use of the system components at all times. Thus it achieves greater over-all production and efficiency. It assumes many functions that were formerly the responsibility of the programmer or operator, such as scheduling, allocation of memory, and assignment of input/output units. Because these functions are performed under central control, changes in schedule, system configuration, and program sizes can be readily accommodated.

It is this ability to control the processing pattern that provides the B 5000 with the potential for smooth growth. Since Processors, Input/Output

Channels, and core Memory Modules are independent of each other, the user may start with a basic system and only enough memory space to process his current programs. Large programs are processed in workable segments, so that the available memory is not filled with one job. As the volume of work swells, or larger programs are used, Memory Modules may be added. This increases Processor speed because more information can be processed concurrently. When input/output requirements exceed the capacity of the original equipment, a new Input/Output Channel can be added. Since the Master Control Program automatically adjusts equipment assignments to use all available units, reprogramming is *never necessary*. Finally, a considerable growth in workload may warrant a second Processor, so that true parallel processing, as well as Multi-Processing, can be performed. The growth of the system in small increments—suited to the workload—is possible because the B 5000 is modular and because the programs are independent of the hardware. This Program Independent Modularity is the ability of the B 5000 to process programs on the equipment available, always making the most effective use of the system configuration, according to the needs of the application.

SECTION 2

PROGRAMMING LANGUAGE

ADVANTAGES OF PROBLEM-ORIENTED LANGUAGES

This section explains the language used by the programmer to communicate with the B 5000 System. Three advantages of this language, as compared to conventional machine and compiler languages, are discussed:

1. Reduced programming time.
2. Reduced time to compile and run the object program.
3. Simplified debugging and program maintenance.

Programming for the B 5000 System is done in a problem-oriented language, rather than machine language. Thus the programmer is free to concentrate on the job of finding a logical solution and recording that solution in a language akin to his own.

Since most problems can be placed in one of two categories, computation or data manipulation, two distinct problem languages have been made available to the B 5000 programmer, one for each problem type.

COMPUTATIONAL LANGUAGE (ALGOL)

The person responsible for solving a mathematical problem thinks about his subject in algebraic terminology. Many algebraic languages have been devised and have been in use for several years. However, experienced and capable computer users have become increasingly concerned in recent years about the number of these languages being produced. It has seemed that each new machine was accompanied by the announcement of a new automatic programming language.

An international conference of computer specialists was held to solve this problem. At this conference the groundwork was laid for development of a single language which would have three major characteristics:

1. It was to be independent of any computer so that programs written in this language would be available to all users regardless of the machine used.
2. It was to be as close to the thinking language of the programmer as possible to shorten the time required to formulate the solution on paper.
3. It was to be a language which incorporated every means conceivably needed for expressing solutions to problems of an algebraic nature.

A preliminary report on this conference was issued in December, 1958. The language was given the name ALGOL (ALGOrithmic Language). During 1959 the preliminary ALGOL specifications were studied by interested parties throughout the world. Informal meetings were held and correspondence was carried on by many computer-user groups in an effort to perfect this language.

Early in 1960, another international conference was held in Paris. As a result, ALGOL 60 was born. BURROUGHS has had a part in the development of ALGOL and is proud to include it as a part of the B 5000 Programming Language.

The following brief example may give the reader an idea of the use of ALGOL. A more complete explanation of ALGOL with a more complex example is given in Appendix A.

It is desired to add the quantities of A and B, and, if the sum (C) is greater than zero, stop the operation. When coding in ALGOL 60, this operation can be expressed in one simple statement, or program step, and that statement is like the thought process of the programmer.

IF A + B > 0 **THEN GO TO** HALT;

Also, in preparing this statement for input, it would be keypunched as it appears.

Compare this to the steps required when the same problem is coded in the machine language of another computer. It would be keypunched as follows:

```
0 0000 10 1000
0 0000 12 1001
0 0000 18 2000
0 0000 34 2050
```

DATA-PROCESSING LANGUAGE (COBOL)

For defining a data-manipulation problem it would be ideal to use a language which is native to the pro-

grammer. Therefore, it has been a goal of the designers of automatic programming systems to provide as many English words as possible in the programming languages designed for business.

The same problems, however, that confronted scientific users became apparent in data-processing operations: too many languages were being developed and put into use. In May, 1959, a meeting was held in Washington with representatives from industry, government, and computer manufacturers in attendance. They agreed that the development of one common language tailored for business use was both desirable and feasible. There were three major requirements:

1. The need to translate existing data-processing problem solutions efficiently from one type of computer to another with minimum conversion costs.
2. The need for program documentation in a form allowing changes and additions with minimum time and expense.
3. The need for reducing the training period for programming personnel.

A report outlining initial specifications of a COmmon Business Oriented Language (COBOL) was published in April, 1960. These specifications represent COBOL 60.

On February 4, 1961, the COBOL Maintenance Committee—a group of 12 ADPM manufacturers, including Burroughs, and 10 interested industrial users—will complete the newest revision to the COBOL Specifications. The revised specifications will represent COBOL 61, which will be a programming language for the B 5000. The B 5000 COBOL translator is designed so that future modifications such as COBOL 62 can be readily incorporated.

With COBOL, as with any language, certain rules of construction must be followed. The features of COBOL are outlined in Appendix B, together with an example. However, to give an indication of the nature of COBOL, a simple example is given here.

It is desired to update an employee's annual FICA immediately if his monthly FICA is greater than or equal to the average. If it is less than the average, it is to be given special consideration before updating the annual balance.

Using COBOL, this situation can be handled by one procedure, and would be handled as follows:

IF MONTHLY-FICA LESS THAN 16.00 GO TO SPECIAL-FICA; OTHERWISE ADD MONTHLY-FICA TO ANNUAL FICA.

Using the machine language of another computer, the same situation would be coded as follows:

```
0 0000 10 1000
0 0000 18 2000
0 0001 34 2050
0 0000 19 1010
```

The BURROUGHS B 5000 Programming Language, therefore, includes the two most widely accepted computer languages yet devised, ALGOL and COBOL. They are usable in the form designated by their originators and are not just another manufacturer's modification. The use of ALGOL and COBOL will greatly reduce the time required to program the problem and will result in more B 5000 production per hour. Extensive studies, comparing the time required to code a variety of problems in machine language with that to program them using ALGOL, have shown a reduction in the over-all time in a ratio of approximately 20 to 1.

REDUCED COMPILING AND RUNNING TIME

Another important factor contributing to the Through-Put of the B 5000 is the greatly increased compilation speed of its translators. Programs that once required machine compilation time from 30 minutes to nearly two hours can be handled by the B 5000 in 30 seconds to a minute or two.

Not only does the B 5000 translate fast, but it produces a very efficient object program. In the past, an experienced programmer could see at a glance many superfluous instructions in a compiled program. This is not true of a B 5000 object program.

SIMPLIFIED DEBUGGING AND PROGRAM MAINTENANCE

A serious obstacle to the completion of a program is often the amount of time required for debugging. The B 5000 programmer need not concern himself with the easily made and hard-to-find errors created in the manual process of translating a logical solution into detailed machine instructions. He looks only at the logic of the problem, which is defined in an understandable language.

The difference in complexity of debugging requirements can best be illustrated by an example. Suppose that a reporter who does not understand Japanese is required to write a story for Japanese distribution. Upon completion of the task a Japanese proofreader informs him only that the story is not correct. The reporter would then be required to check each character of his story with the aid of an English-Japanese dictionary, a task which could prove as time consuming as the writing itself.

On the other hand, suppose that an interpreter were available to inform the reporter of factual errors in the story due to the language difference. The reporter then need only review his English copy to correct the specified facts.

The B 5000 incorporates a translator analogous to the interpreter. The programmer need never know computer language to write an effective program for it.

Program revisions can be as tedious as debugging. Such revisions may be dictated by an altered problem definition, often occurring many weeks after completion of debugging. To facilitate making these changes it is desirable to document the program with logical flow charts, file layouts, and solutions to the problem. With the B 5000, the documentation is readily understandable—not only to the programmer but to anyone concerned with the problem. Special-purpose flow charts slanted toward particular machine characteristics or the machine language instructions themselves are not required. Occasional changes to the program are accomplished in problem-oriented language. The changes are concerned with the logical solution, not the machine language instructions.

PROGRAMMING ADVANTAGES OF THE B5000 SYSTEM

REDUCED PROGRAMMING TIME

The BURROUGHS B 5000 is designed to make use of the best programming language presently available, a problem-oriented language which includes the most recent developments in this field, ALGOL 60 and COBOL. Because ALGOL and COBOL were devised by top people in their respective fields, it is reasonable to expect that these languages will not be replaced for years to come. ALGOL and COBOL have already received wide acceptance and their use is fast becoming universal. It is therefore possible to create a program library which will have a life span beyond that of the particular computer equipment being used at the time. The exchange of programs between different users is also practical.

The design characteristics of the B 5000 contribute significantly to greatly reduce total programming time. The programmer need not be concerned with such things as actual memory locations, specific input and output unit and magnetic tape unit designation. In the B5000, these details are handled automatically while the program is being run. The same object program can be run without recompilation even when the system configuration changes. These housekeeping details are automatically handled by the Master Control Program.

REDUCED COMPILING AND RUNNING TIME

The BURROUGHS B 5000 is capable of very fast compilation speeds and efficient object programs because the system has been designed to take advantage of the latest automatic programming techniques.

Recent BURROUGHS compilers have made use of these techniques, resulting in compilation speeds as much as 50 times faster than those of other compilers.

Conventional Compiler Functions. The operation of compilers must be considered in order to understand the advantages of these BURROUGHS compiler techniques. A compiler is a program able to translate from one language to another, i.e., from a language which is intelligible to a programmer to a language which is understood by an electronic computer. Programming languages have changed and have begun to approach the thinking language of the programmer. These improvements, however, have resulted in the programming language looking less and less like the machine language. This means that the compiler requires more and more time to perform the translation functions required to produce an object program in machine language.

The conventional compiler does three basic things:

1. Examines the programmer's language.
2. Creates an intermediate language.
3. Produces machine language.

First, it examines the programmer's language (source program), one character at a time and establishes a dictionary of separate entities, such as identifiers, numbers, and variables. The compiler is able to do this by recognizing separator and operator symbols of the source language such as colons, equal signs, and parentheses. For example, if the programmer had written somewhere in his program:

$$\text{START:}\quad A \leftarrow 7 \times (B + C)$$

the compiler would begin to scan with the letter S. It would then encounter the letters T, A, R, and T in that order. When it recognizes the colon as the next character, the compiler knows that START is a separate item and would add it to the list. Next the compiler encounters A followed by an arrow, so it would add A to the list. Similarly for 7 and B and C. The dictionary contains, in addition to the item's name, its type and an indication of its location.

Secondly, the compiler produces an intermediate language which essentially breaks up the program into separate operational groupings. For instance, in the above example, the following equation must be evaluated:

$$A = 7\ (B + C)$$

Earlier conventional compilers would break this equation up into two operations:

$$A = 7 \times T$$
$$T = B + C$$

(where T stands for some temporary storage location)

A temporary location is established each time an enclosure symbol is encountered during the scanning process. When a left parenthesis is found in an arithmetic expression, for example, it is immediately known that a single value is to be defined. The set of operations for computing this value will be terminated by the associated right parenthesis. When the termination will occur is still unknown. Enclosure symbols may be nested and expressions quite complex. Therefore, this problem has been handled by creating many temporary locations. Later an attempt must be made to eliminate as many as possible and the intermediate language condensed.

Finally, the actual machine language is produced through the use of generators within the compiler itself. Because these generators (subroutines) are of a general nature they must be constructed to take care of all possible cases. Therefore, the resulting machine language (object program) contains unnecessary instructions. For instance, the above example might result in the following:

```
LOAD      B
ADD       C     Add operation generator.
STORE     T
LOAD      7
MULTIPLY  T     Multiply operation generator.
STORE     T
LOAD      T     Store operation generator.
STORE     A
```

An experienced programmer would use fewer instructions to arrive at the same solution:

```
LOAD      B
ADD       C
MULTIPLY  7
STORE     A
```

Therefore, the object program produced by earlier compilers would take, in most cases, considerably longer to run than one written by an experienced programmer.

Burroughs Compiler Techniques. The first task of a compiler, that of scanning the source program, is also a part of the B 5000 translators. It is interesting to note, however, that in a conventional compiler this process is just a first step. On the B 5000, one pass through the programmer's language is all that is required.

The effectiveness of the BURROUGHS compiler technique is most readily recognized in the second phase of compilation. The example A = 7 (B + C) as described illustrates that parentheses caused conventional compilers to do a great deal of extra work in setting up temporary storage locations. Nonetheless, parentheses are needed in the source language to eliminate ambiguities. For instance, the example could be interpreted another way if there were no parentheses. Namely:

$$A = 7B + C$$

This problem of ambiguity was also quite perplexing to philosophers who were considering logical propositions until a Polish logician, J. Lukasiewicz, developed a notational system which did not require enclosure symbols. His scheme is commonly called Polish notation. The BURROUGHS technique produces an intermediate language using Polish notation concepts.

POLISH NOTATION. The essential difference between Polish notation and conventional notation is that operators are written to the right of a pair of operands instead of between them. For example, the conventional B + C would be written BC+ in Polish notation. Looking again at the example, A = 7 (B + C), it could be written as follows:

$$BC + 7 \times A =$$

Any expression written in Polish notation is called a Polish string. In order to fully understand this concept, the rule for evaluating a Polish string should be known.

The rule can be summarized in a few steps:

1. Scan the string from left to right.
2. Remember the operands and the order in which they occur.
3. When an operator is encountered do the following:
   a. Take the two operands which are last in order;
   b. Operate upon them according to the type of operator encountered;
   c. Eliminate these two operands from further consideration;
   d. Remember the result of (b) and consider it as the last operand in order.

Following this rule through the Polish string, BC + 7 × A =, step by step would result in:

SECTION 3

SYSTEM CHARACTERISTICS

GENERAL DESCRIPTION

When a program has been written in one of the languages acceptable to the BURROUGHS B 5000 System—either ALGOL 60 or COBOL 61—it is punched on cards and loaded into the system. The object program is then compiled and recorded on a magnetic tape called the program tape or punched on program cards. The compiled program consists of program segments and an associated Program Reference Table. The length of the program segments varies and is determined by the compiler according to the over-all length and nature of the problem. Program parameters are also produced during compilation. These specify the input-output devices required, the amount of storage required for the program itself and for its associated data, and other information necessary for processing the program. Figure 3-1 shows the transformation of the source program into an object program.

STORAGE DRUM

The BURROUGHS B 5000 System includes a magnetic drum as well as core storage. Part of the drum is reserved for storage of the compiler and the Master Control Program. This area is not available to the user. The remainder of the drum is used for program segments and their associated Program Reference Tables, as illustrated in Figure 3-2.

CORE MEMORY

The Master Control Program is loaded into memory initially by depressing the Program Load button.

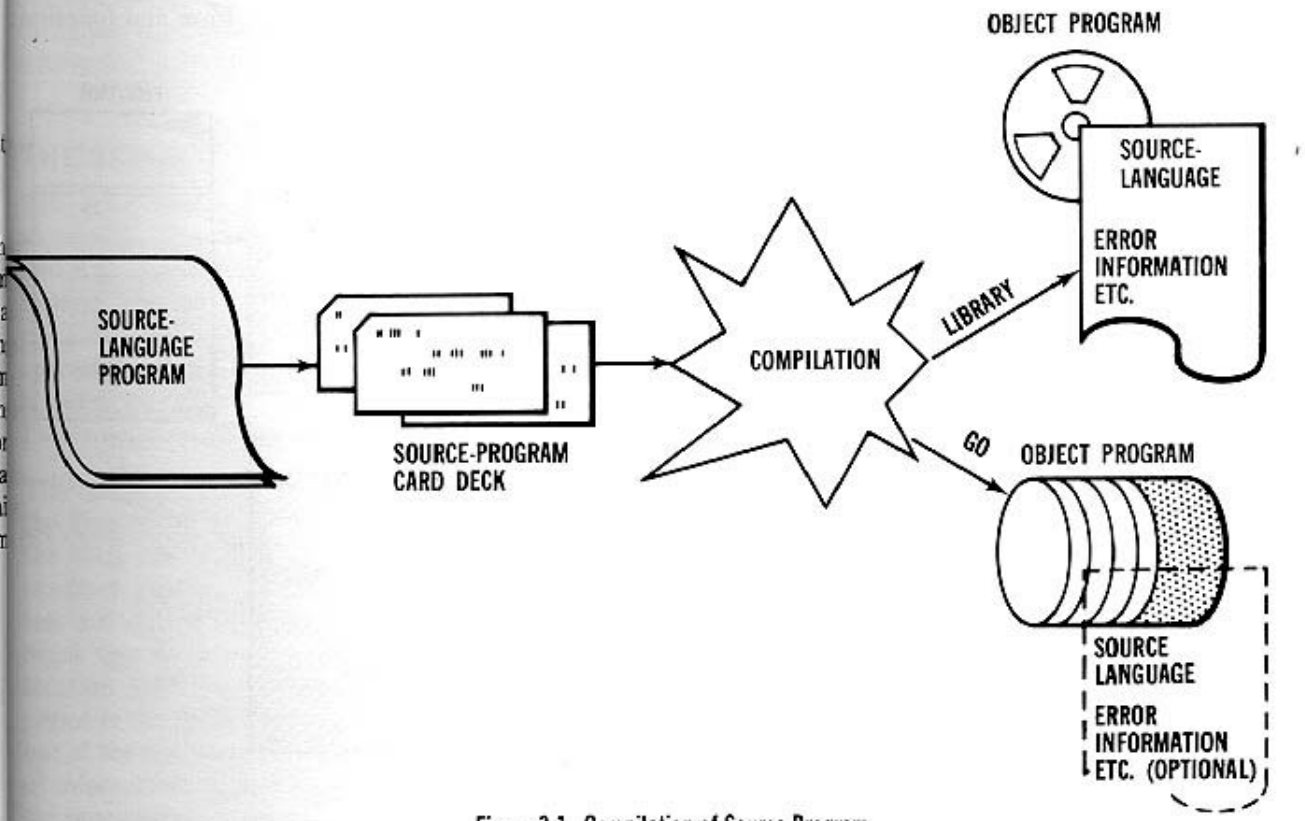


Figure 3-1. Compilation of Source Program

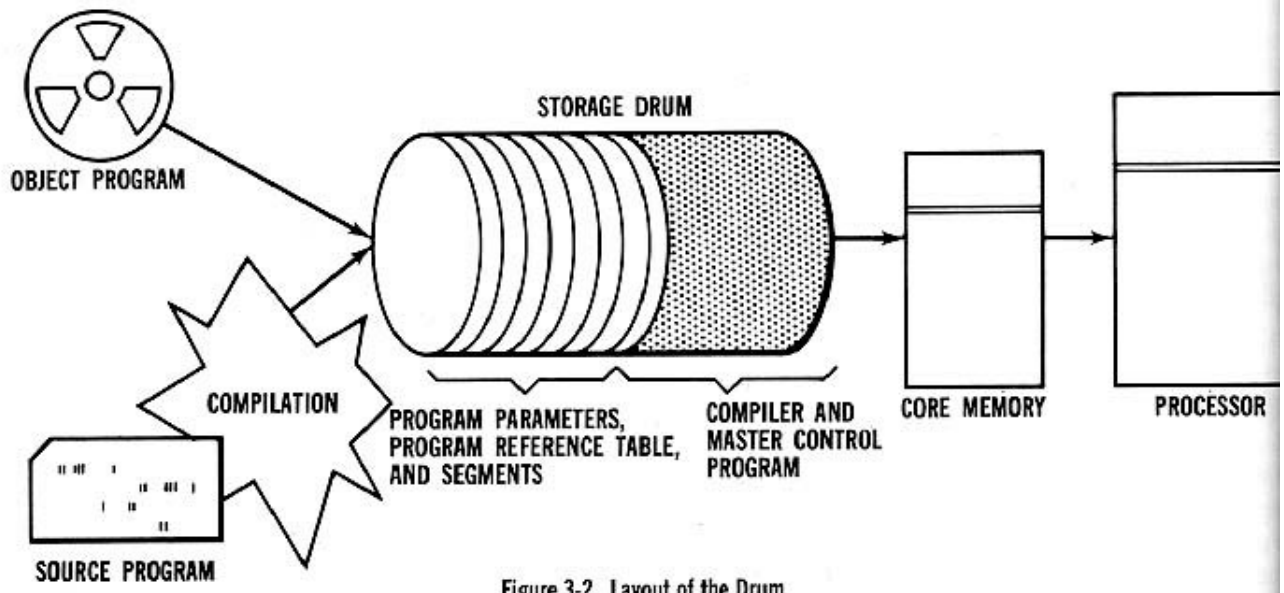


Figure 3-2. Layout of the Drum

Its function is to select the scheduled programs from either the program tape or punched cards and load them onto the drum. Then, on the basis of the program parameters supplied, the Master Control Program allocates memory space for each program to be processed. Next, it loads the required program segments and associated Program Reference Table for each job. At this point the Program Reference Table and program segments pertaining to each job are contained in memory. Other areas of memo have been allocated to contain the Stack and inpu output information, as shown in Figure 3-3.

WORDS

Information is stored in the Program 'Referen Tables and program segments in words containi 48 bits of information. There are three types words, each having a particular form and functio

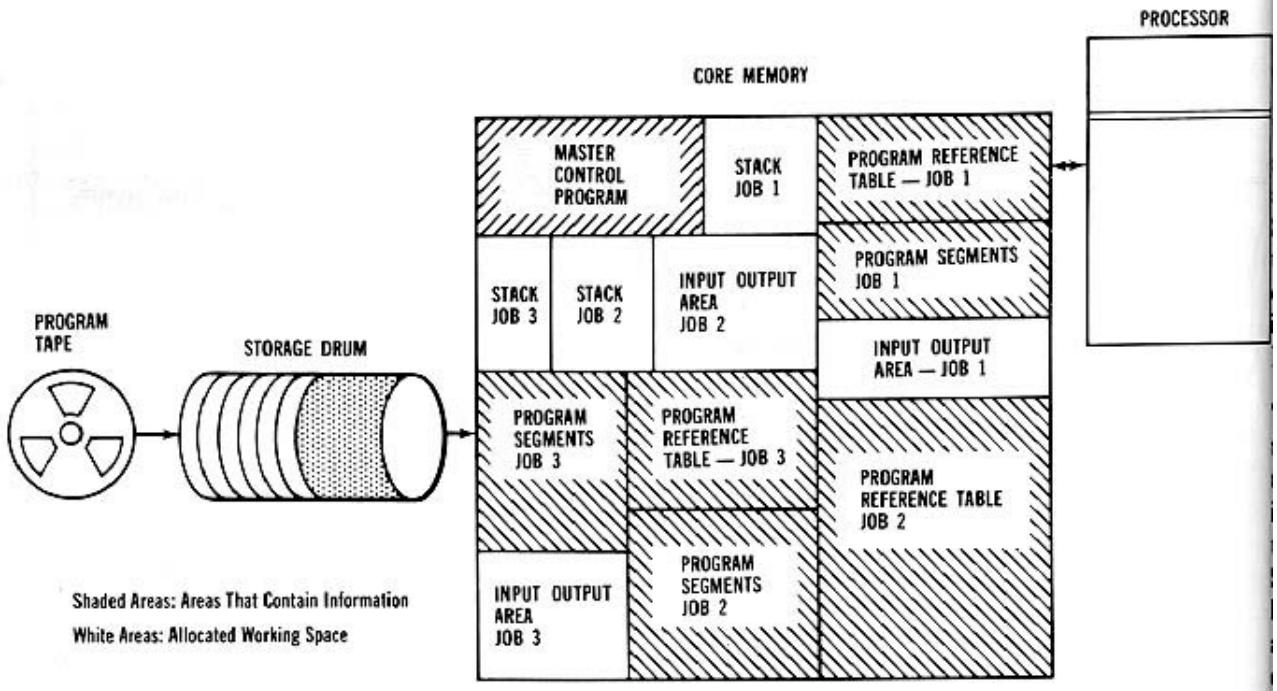


Figure 3-3. Allocation of Core Memory

descriptors, operands, and program words. The descriptors and operands make up the Program Reference Table, and program words are found in the program segments. Each type is introduced below and discussed in further detail.

PROGRAM REFERENCE TABLE

The Program Reference Table is an area of storage containing control words that are used to *locate* data. They may also *describe* the type of input/output operation to be executed. These control words, known as descriptors, provide the base or starting address of a program segment, file of records, input/output area, or subroutine. When they specify input/output operations, they designate the unit to be used, format, and the location and amount of information to be transferred.

Operands are also located in the Program Reference Table and provide direct access to single values. This type of word contains the *actual* value instead of the location of that value.

PROGRAM SEGMENTS

A program segment, composed of program words, is a self-contained group of steps that does a portion of the processing for a job. Each program word is divided into four syllables. And there are four *types* of syllables used in program words: operand-call, descriptor-call, literal, and operator. Essentially these are the instructions for processing the data referenced by the Program Reference Table.

THE STACK

This is an area of memory set aside for temporary storage of information. It is associated with the A and B registers, the arithmetic registers of the B 5000 System. The registers form the first and second locations of the Stack, and information transferred to or from the Stack passes through them. Its operation is much like that of the stack used to compile the Polish notation described in Section 2.

The last piece of information placed in the Stack is the first to be accessed again. Previous entries in the Stack are "pushed down" when new information is added; that is, the contents of the A and B registers are shifted into storage locations allocated for Stack use. As more space is required, a new storage location (addresses are used in ascending order) is added to the Stack, receiving the information shifted out of the registers. Figure 3-4 illustrates the shifting of information as new data is brought to the Stack for processing.

The term "top of the Stack" refers either to the first Stack location in memory or to the A or B register, depending on the placement of the data. If an operand is transferred from the Program Reference Table it is placed in the A register, which is, at that moment, the top of the Stack. The results of many operations, however, are placed in the B register, leaving the A register empty; in this case, the B register becomes the top of the Stack. Under some conditions both the A and B registers are cleared, and then the first memory location in use is the top of the Stack. Whenever a sequence of operations requires information stored in the Stack memory locations, it is automatically shifted into the registers where it can be processed.

The location currently in use as the top of the Stack in memory is recorded in the S register. If new locations are added to hold information brought to the Stack, the S register is counted up. When an operation causes items in the Stack to be removed, those locations are removed from the Stack until needed again, and the S register setting is counted down by the corresponding amount. The total number of locations used varies according to the processing sequence. Referring to Figure 3-4, assume that a series of operations uses the information stored in the A and B registers and in Stack locations 1002 and 1001 (stage 3 in the example). The result of the operations is held in the B register, and the S register is set to location 1000. Figure 3-5 shows the condition of the Stack at this point. (Note that the A register is empty, and therefore the B register has become the top of the Stack; if both registers were empty, location 1000 would be the top of the Stack.)

The allocation of memory in Figure 3-3 shows three areas set aside as Stacks for Multi-Processing of three different jobs. Since the Stack is merely a flexible working area, its actual location is unimportant. The Stack in use at any given moment is associated with the A and B registers, and the memory address at the top of the Stack being used is recorded in the S register. When the Processor switches from one job to another, any information held in the A and B registers is automatically shifted into memory locations prior to transfer of control. The current S register setting is always preserved for re-entry to the job.

The Master Control Program, described in Section 4, controls the processing sequence of various jobs. It allocates Stack memory space according to the number and size of programs in memory at one time. Switching from job to job, and therefore from Stack to Stack, is also supervised by the Master Control Program.

SECTION 4

MASTER CONTROL PROGRAM

INTRODUCTION

The Master Control Program consists of a special set of routines designed by the manufacturer to provide automatic control over scheduling, allocation of memory, input/output operations, assignment of hardware functions, multi- and parallel processing, and all interruptions of system operation. The Master Control Program (MCP) is permanently recorded on Storage Drum 1 by the manufacturer. Whenever it is being used, the Processor operates in a special mode—the Control mode.

The features and purpose of the B 5000 Master Control Program are explained in this section. Operation during the Control mode and the functions of each of the routines which constitute the Master Control Program are described.

Computers have been developed to save time and to reduce the chance of errors. By providing facilities for performing extremely complex computations and for eliminating tedious clerical functions, they enable users to make creative use of masses of information. Still, operator intervention in computer functions is time consuming and introduces the chance of error. One reason the B 5000 is an advanced system is that it incorporates the means for fully automatic operation. The features of the MCP encompass three general areas: computer functions, automatic system assignment and coordination, and Multi-Processing.

COMPUTER FUNCTIONS

The MCP controls all computer functions except physical tape and card handling. It provides supervisory control instructions to the operator so that he can make major decisions in directing the processing, but provides the means for automatic handling of all but the most unusual situations. The operator is always able to communicate with the system but seldom needs to. The MCP can analyze error conditions and provide an appropriate course of program action if the object program does not specify one.

AUTOMATIC SYSTEM ASSIGNMENT AND COORDINATION

Certain operations, which were the responsibility of the programmer with other computer systems, are performed automatically with the B 5000. By sensing the system's environment, the MCP is able to assign memory areas to program segments, input/output areas, and Program Reference Table entries. Input and output files are automatically identified and given physical unit designations. Standard tape-handling procedures provide for labeling, end-of-tape, and end-of-file conditions.

MULTI-PROCESSING

The MCP can determine the optimum sequence and combination for processing a batch of jobs. When new or higher priority jobs are introduced, it can adjust its schedule and reorganize the system environment to accommodate the new work. The MCP allocates memory to gain maximum efficiency in processing, and reallocates whenever necessary to preserve optimum processing of a reordered job sequence. The MCP also controls the execution of individual program segments and initiates all input/output operations. Because of its ability to oversee all the programs being processed as well as the individual segments, it can keep input/output devices operating at maximum efficiency with little sacrifice of time.

ELEMENTS OF THE MASTER CONTROL PROGRAM

The MCP includes a group of special-purpose routines that anticipate and provide for almost all operating circumstances.

1. The Executive Routine coordinates the operation of the MCP by determining what type of control function is required and transferring control to the proper routine. It also supervises input/output operations and the execution of all program segments in memory. When processing is interrupted, it determines the cause. If necessary, the operator is notified; otherwise the Executive Routine takes action either to rectify the situation or to continue processing on another program segment.

2. The Schedule Routine evaluates the priority and equipment requirements of a batch of programs. It schedules these and, if necessary, reschedules to maintain efficient and continuous processing.
3. The Environment Control Routine allocates memory space and input/output devices according to the specifications provided by the Schedule Routine.
4. The Exceptional Condition Routines provide standard error-handling procedures.

ENTRY TO THE CONTROL MODE

The Processor transfers to the Control mode whenever processing is interrupted. Any circumstance that stops processing is called an interrupt condition. It may be caused by operator communication with the system, by developments in the program being executed, or by the hardware.

When an interrupt condition is encountered, control is transferred from the object program to the MCP after execution of the program syllable is complete. The contents of the arithmetic and control registers are stored in Stack locations, and the highest address of these locations is placed in the top of the Stack so that return to the object program can be made. The Executive Routine determines the cause of the interrupt and then transfers control to the appropriate portion of the MCP for handling of the condition. When the interrupt condition has been satisfied, registers are restored and control is transferred to an object program.

THE INTERRUPT CONCEPT

The term "interrupt condition" is used in a special sense with the B 5000 System. It does not imply that work is interrupted or that the system is held up in any way. Rather, a transfer of control is taking place, and the MCP may initiate certain types of operations that can proceed simultaneously with computation. Input/output operations, for example, have usually been controlled by the individual program in process. In the B 5000 System, input/output operations are part of a centralized communications control, and cause an interrupt condition each time they are executed. Since the relatively slow speed of input/output devices normally causes a certain amount of idleness in the Processor, especially for business applications, the B 5000 has been designed to permit a maximum use of all peripheral devices. The minute amount of processing time that is lost through interrupt conditions is far outweighed by the over-all increase in production. Each time an interrupt condition indicates to the MCP that an Input/Output Channel is free, processing paus momentarily so that the MCP can initiate new o erations to make full use of all the system co ponents. In summary, then, interrupt conditio provide opportunities for the MCP to transfer co trol to the area of processing which will use t equipment most effectively, or for the MCP to r spond to any error signals, operator messages, unexpected processing conditions. Because it h over-all control of the system, the MCP is able make the most of the available system and to ke the jobs moving.

In the following pages, the parts of the Master Co trol Program are examined in detail. The functio of each routine and its relationship to other routine particularly to the Executive Routine, are outline

EXECUTIVE ROUTINE

The Executive Routine is loaded initially from t drum when the Program Load button is depress and is retained in memory throughout processin It loads other portions of the MCP from the dru as they are needed. As the coordinating member the MCP, the Executive Routine has six ba functions:

1. To initiate all input/output operations;
2. To analyze all interrupt conditions and provi an appropriate course of action;
3. To maintain control transfer points for pr gram segments;
4. To control the use of other routines in the MC
5. To maintain an operations log;
6. To maintain an internal physical system d scription.

INPUT/OUTPUT OPERATIONS

The Executive Routine, in order to initiate a coordinate all input/output operations, maintai several tables in memory. The base locations of t input/output descriptors for all programs in memo are recorded in one of these tables. A second ta keeps a tally of the total number of such descripto in each Program Reference Table. Constant acce to the base location and the total number of each s of descriptors enables the Executive Routine to r erence a particular descriptor when an input/outp operation is to be executed. A third table contain record of the descriptors currently being proces by each Input/Output Channel. Thus the Executi Routine can evaluate the status of any descriptor any time. In summary, the Executive Routine h constantly updated information on the location a status of all input/output descriptors.

SECTION 5

COMPONENTS

INTRODUCTION

The system chart in Figure 5-1 illustrates the remarkable flexibility of the B 5000 Information Processing System. This flexibility is called Program Independent Modularity because the configuration of the system can be altered without causing disruption in processing or requiring program modification. Three types of modules make up a system: Processors, Input/Output Channels, and Memory Modules. These may be coordinated into a system tailor-made for a user's requirements. A system may incorporate one or two Processors, as many as four Input/Output Channels, and as many as eight Memory Modules. Each Input/Output Channel can communicate with any Memory Module, and each Processor can also communicate with any Memory Module. This basic network of communication is augmented by the ability of each Input/Output Channel to control information flow for any of the 26 input-output units, such as Magnetic Tape Units, Storage Drums, Card Readers, Line Printers, etc., that can be included in a maximum system.

The maximum and minimum system configurations are itemized in Table 5-1.

Table 5-1. System Configuration Limits

COMPONENT	MINIMUM	MAXIMUM
Console	1	1
Processor	1	2
Memory Module	1	8
Input/Output Channel	1	4
Storage Drum	1	2
Magnetic Tape Unit	0	16
Line Printer	0	2
Card Reader	1	2
Card Punch	1	1
Plotter	0	1
Keyboard	1	1
Message Printer	1	1
Input/Output Exchange	1	1
Memory Exchange	1	1

Each component is described in the following paragraphs. The Processor, which has been discussed extensively in Section 3, System Characteristics, is not described further here.

CONSOLE

The Console, shown in Figure 5-2, is the operations center of the B 5000 system. It has control switches and indicator lights which provide the operator with a convenient supervisory center. Next to and considered part of the Console control center are the Message Printer and Keyboard, the communication links between the system and operator.

The Console also contains the system switching facilities, the interrupt circuitry, the system synchronization equipment, the power controls, and the interval timer or clock. The interval timer provides a constant interval time interrupt for control purposes. This timer allows the logging of running time of all programs.

MEMORY MODULE

The high-speed Memory Module is the primary storage unit for programs and data. It also couples the computation facility of the Processors with the input/output operations. It is able to accept and transmit data independently of the Processor's activity. In addition, the Processor can access a Memory Module, even though the module is currently communicating with input/output devices.

Each module can store 4096 words. A single system may have from one to eight modules, each independent of the others. This independence facilitates multiple and parallel processing. When a system has one Memory Module, for example, multiple Processor and input/output operations are time-shared in memory. Processor operations can resume as soon as the input/output operation has been *initiated*, so there is virtually no delay. If two or more Memory Modules are available to the system, parallel input/output and Processor operations can be accomplished with no delay.

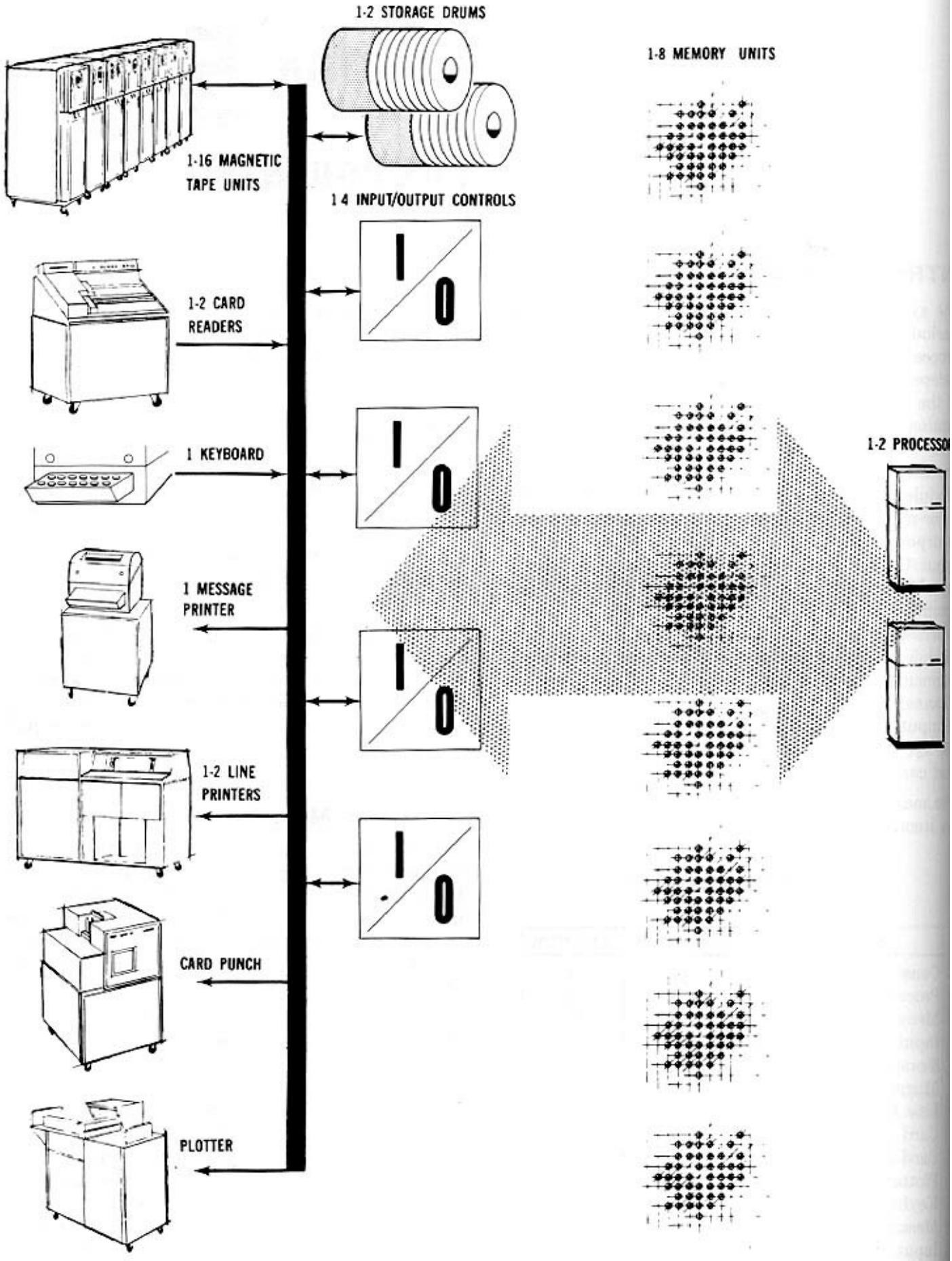


Figure 5-1. System Configuration Chart

Figure 5-2. The Console

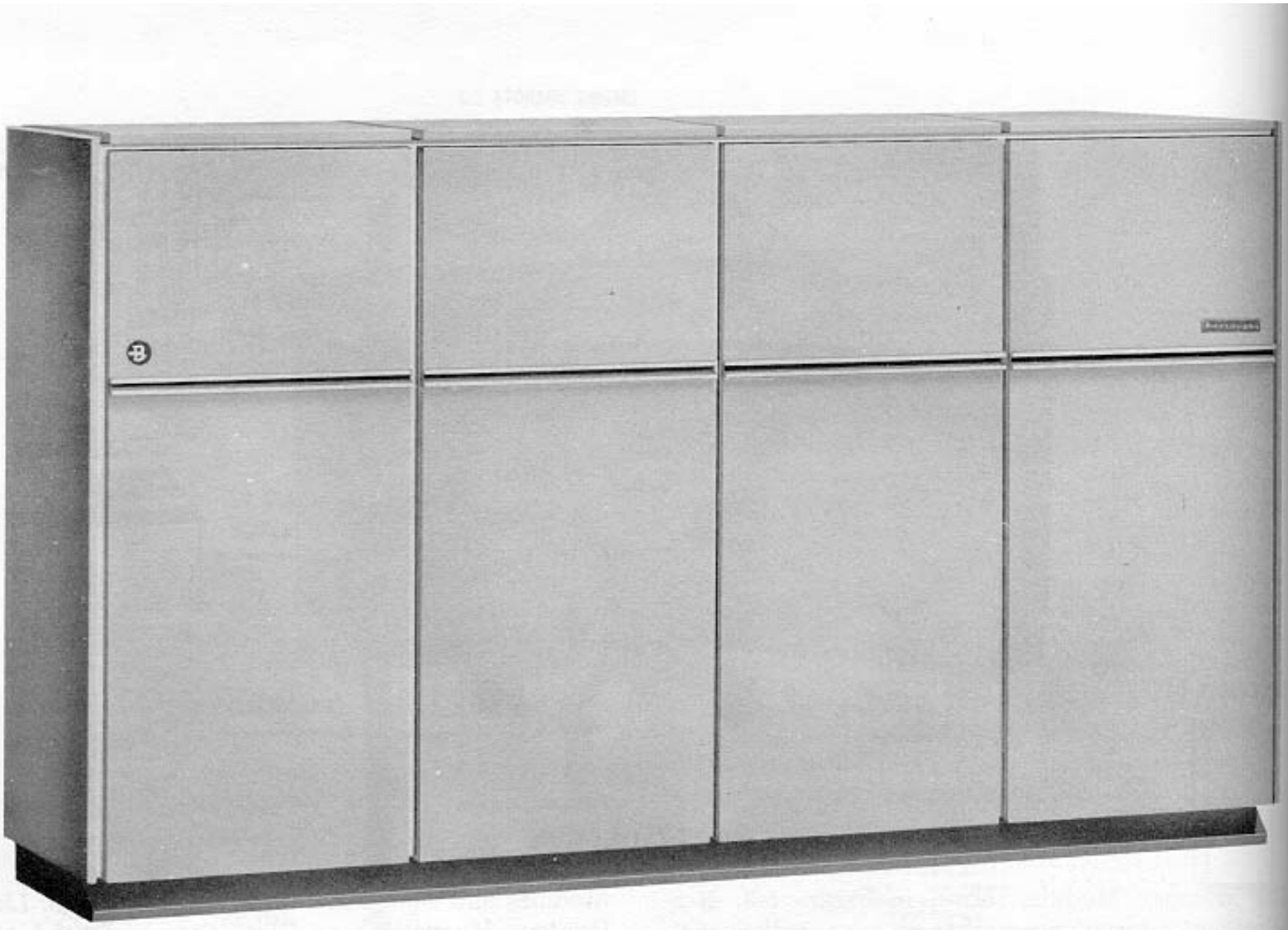


Figure 5-3. Memory Modules and Input/Output Channels

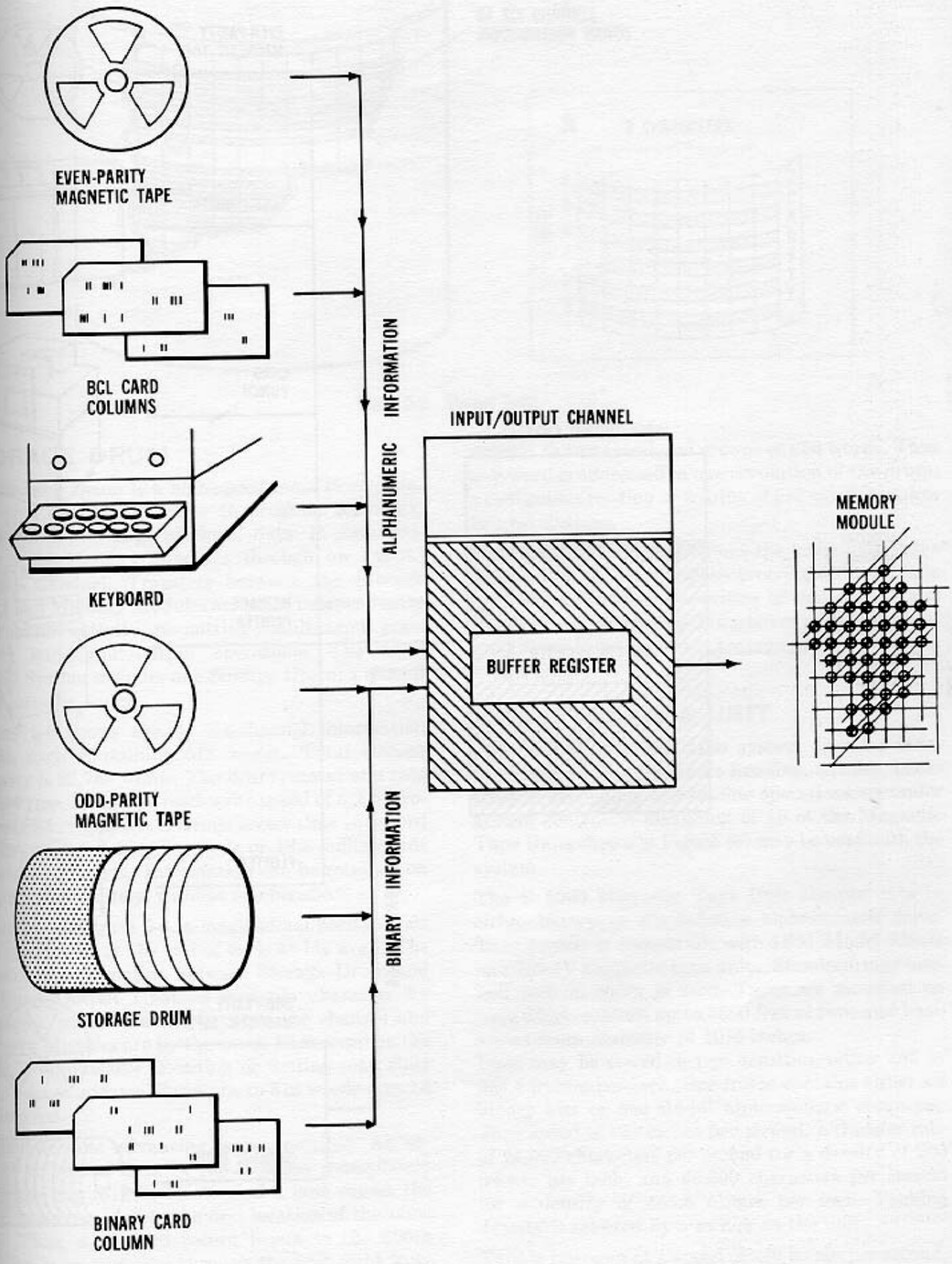


Figure 5-4. Input Information Flow

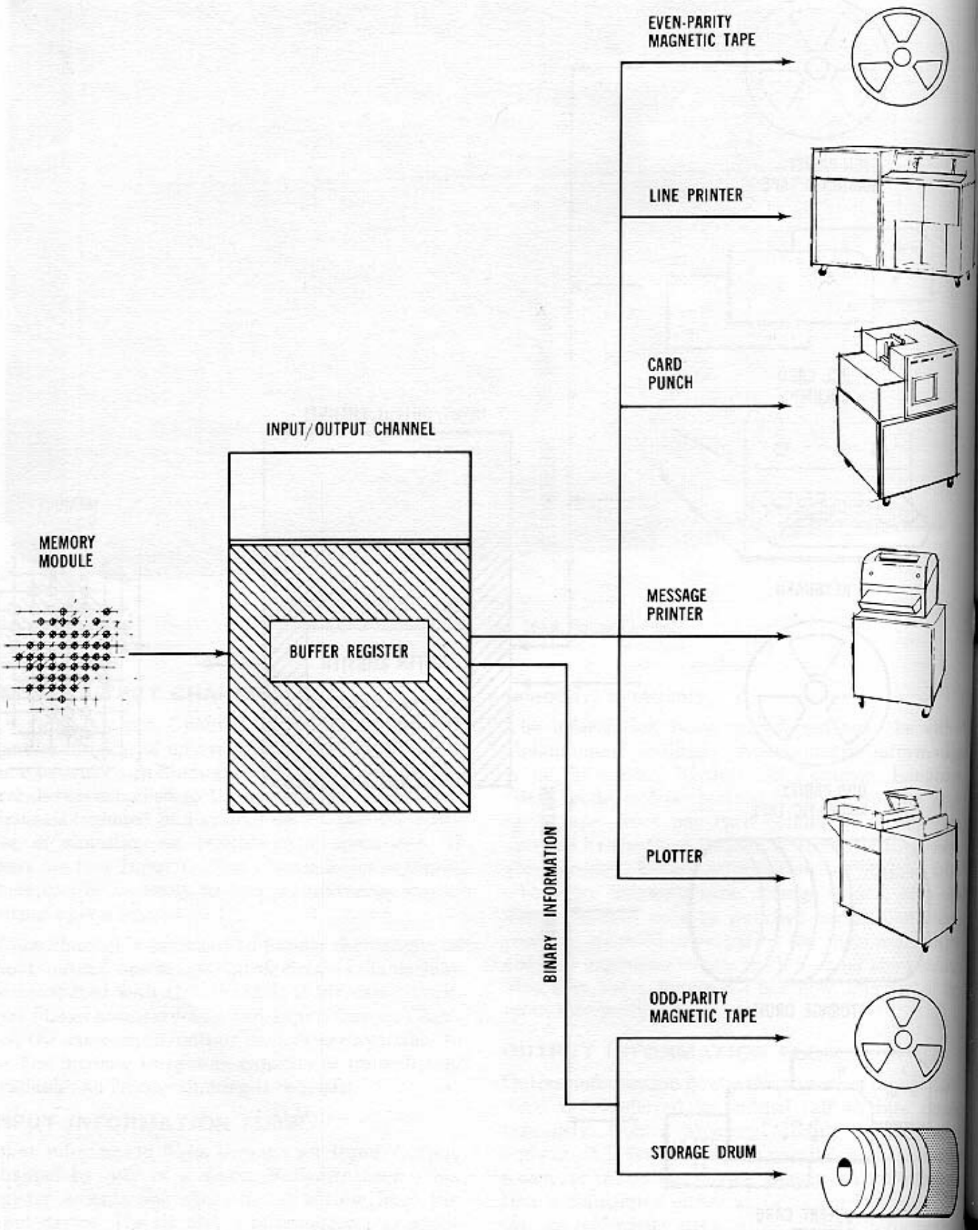


Figure 5-5. Output Information Flow

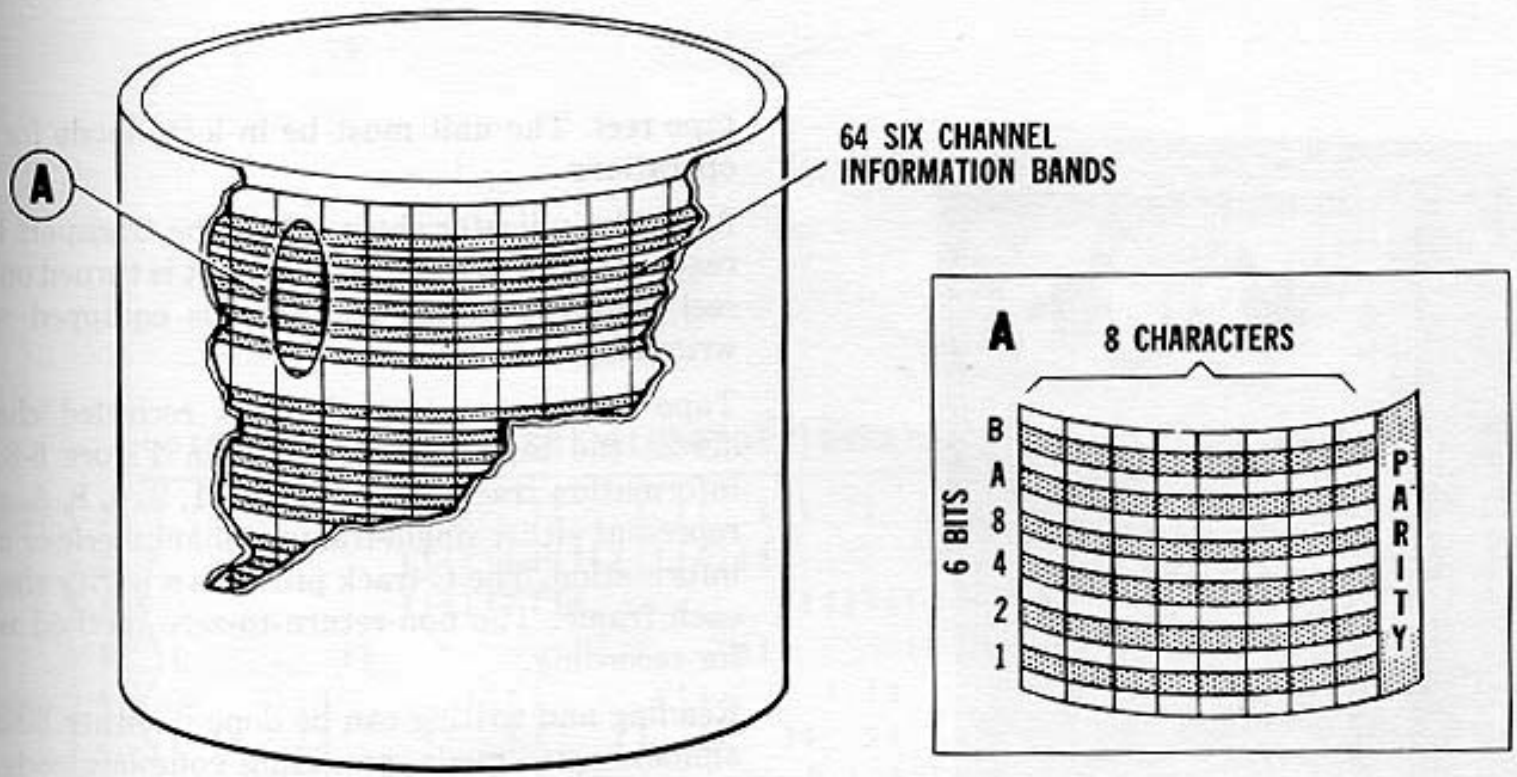


Figure 5-6. Storage Drum

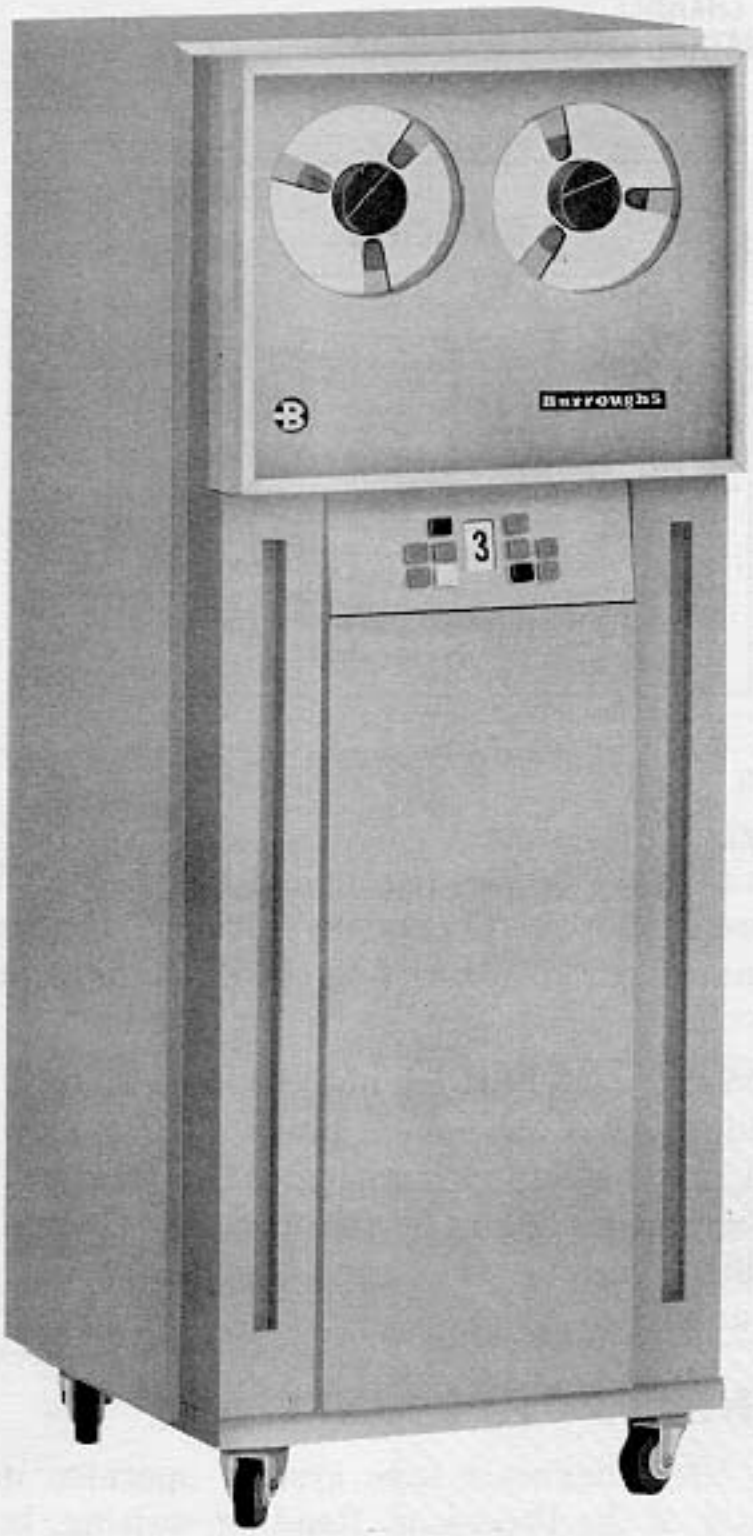


Figure 5-7. Magnetic Tape Unit

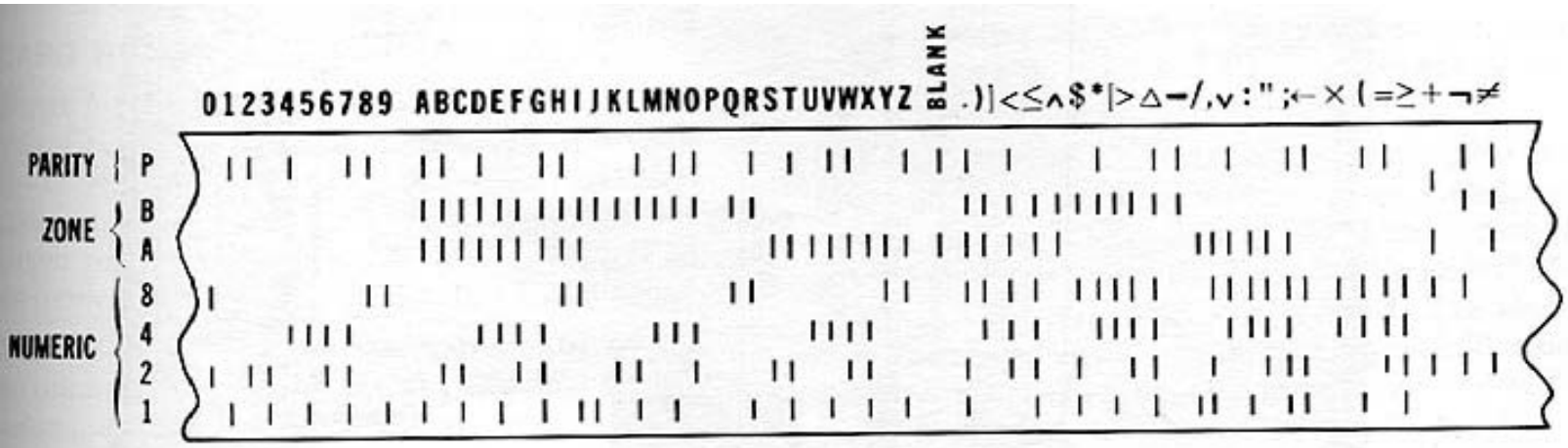


Figure 5-8. Magnetic Tape Format

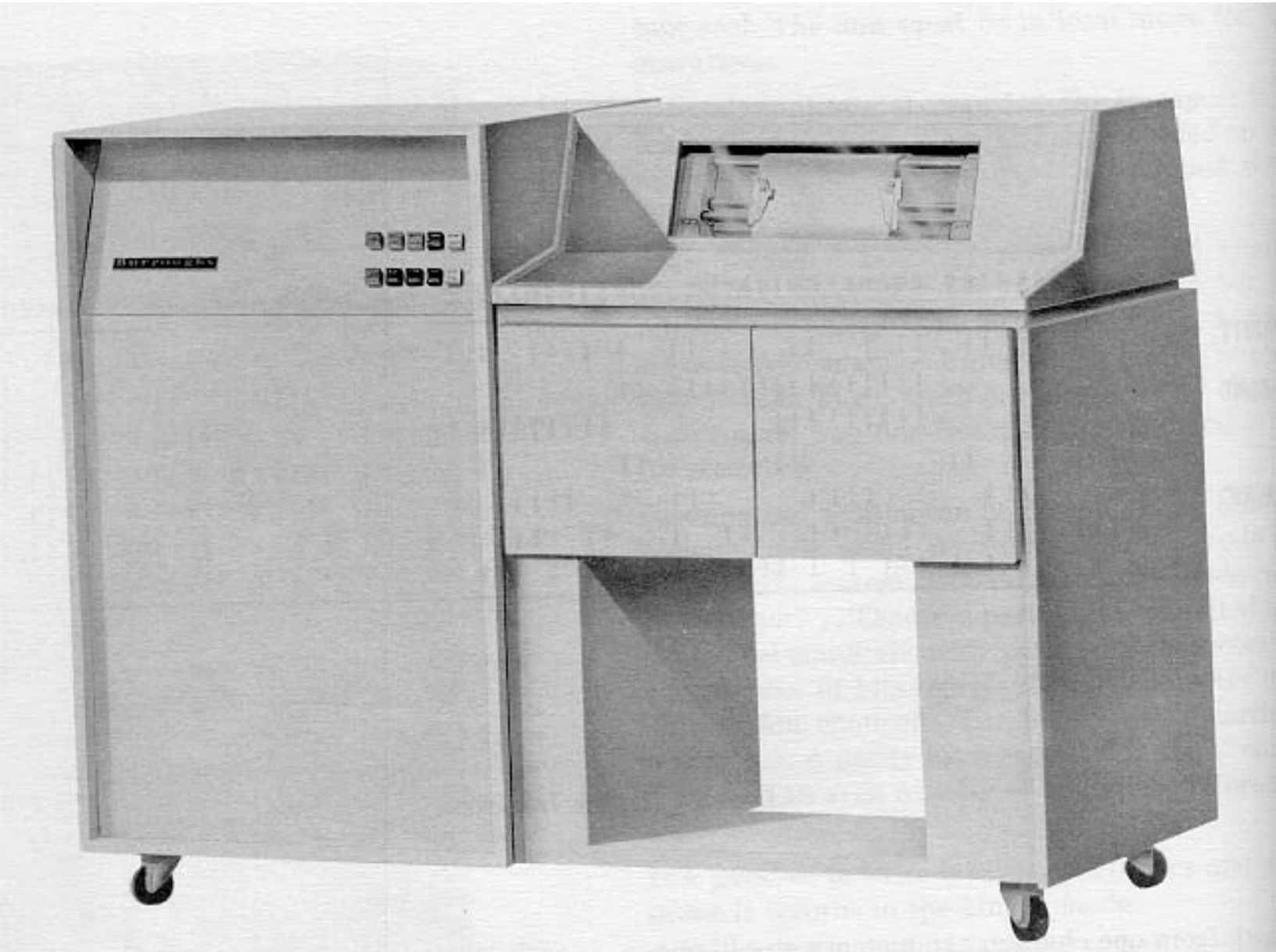


Figure 5-9. Line Printer

Figure 5-10. 800 CPM Reader and Program Card Reader

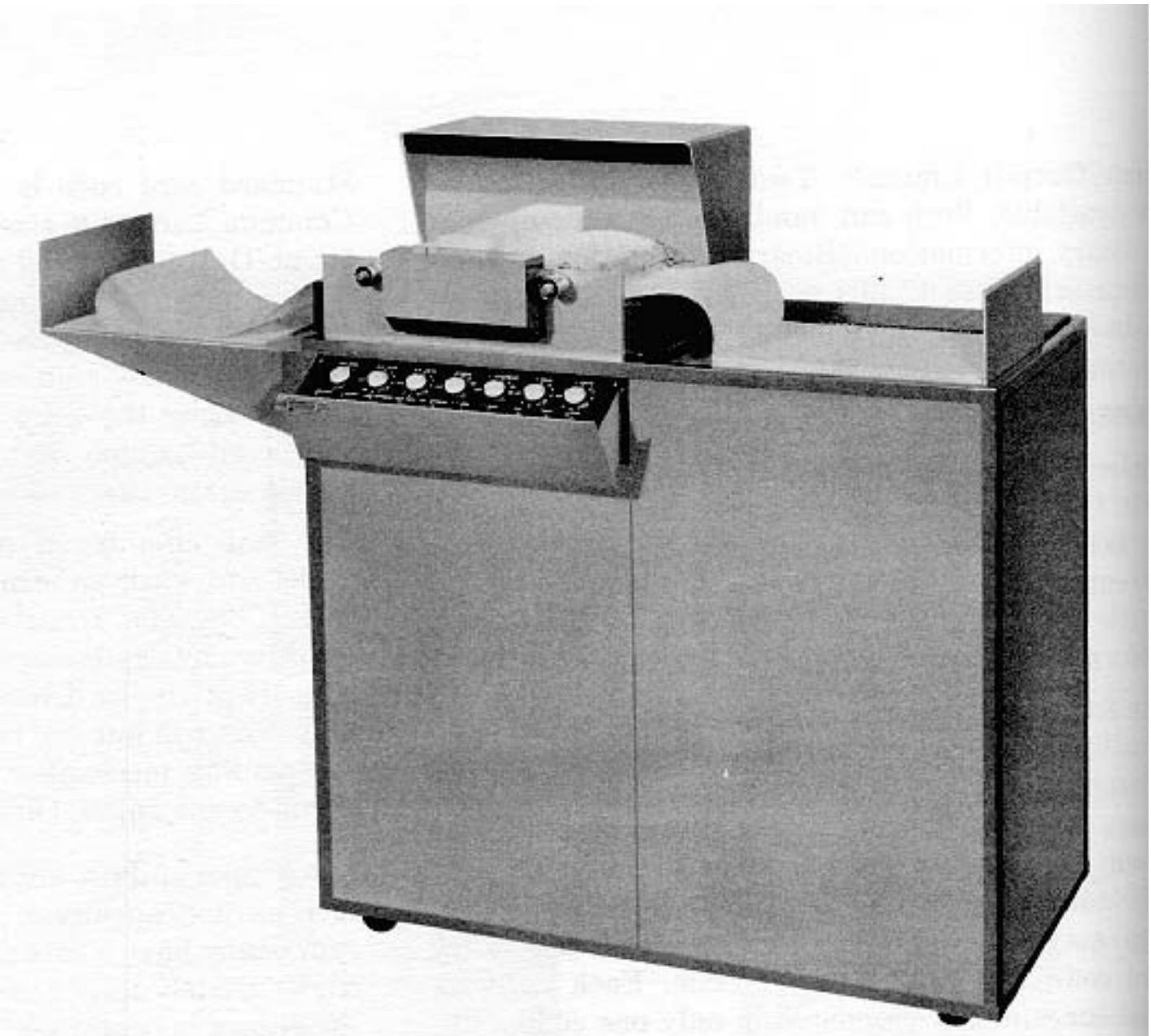


Figure 5-12. Plotter

Figure 5-11. Card Punch

Figure 5-13. Message Printer/Keyboard

APPENDIX A

ALGOL CHARACTERISTICS

INTRODUCTION

For problems essentially computational or mathematical in nature, the language of the B 5000 contains ALGOL 60 with minor restrictions. ALGOL 60, like any language, is formulated from a set of basic symbols and words. These symbols and words are then used to formulate expressions, which in turn are compounded into the programming units of the language called statements. The statements, supported by declarations, are computing instructions to the B 5000 and its compiler.

The purpose of this appendix is to define the terminology of ALGOL 60, showing its efficiency as a programming tool. It is not intended as a computational programming primer.

BASIC SYMBOLS AND WORDS

The alphabet of ALGOL 60 comprises basic symbols (letters, digits, operators, etc.) and some English words. The words of ALGOL 60 are the identifiers, numbers, and strings which are formed from the alphabet.

LETTERS

The 26 capital letters of the English language are used to form identifiers (names or labels for variables, switches, etc.) and strings (special groups of basic symbols):

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

DIGITS

The ten Arabic digits (0-9) are used, both to form numbers and in arbitrary combinations with alphabetic characters to make up identifiers and strings.

OPERATORS AND SYMBOLS

The following arithmetic operators are acceptable.

+	addition
−	subtraction
×	multiplication
/	division
DIV	division (both operands of type **INTEGER**)
*	exponentiation

The relational operators are:

<	less than
≤	less than or equal to
=	equal
≥	greater than or equal to
>	greater than
≠	not equal

The logical (Boolean) operators are:

EQV	logical equivalence (equivalent)
IMP	logical implication (implies)
∨	logical sum (or)
∧	logical product (and)
¬	logical negation (not)

The sequential operators are: **GO TO**, **IF**, **THEN**, **ELSE**, **FOR**, and **DO**.

The separator symbols are:

,	comma; separates arguments of a function, separates elements of a list, subscripts of an array, etc.
.	decimal point
:	colon; follows a label
;	semicolon; separates statements
←	replace by

plus **STEP**, **UNTIL**, **WHILE**, and **COMMENT**.

The bracket symbols are:

( )	Parentheses are used, for example, to enclose parameters of a procedure.
[ ]	Brackets are used, for example, to enclose subscripts of an array.
"	Quotation marks are used to enclose the symbols of a string.

The reserved words **BEGIN** and **END** are also used.

The declarator symbols are: **OWN**, **BOOLEAN**, **INTEGER**, **REAL**, **ARRAY**, **SWITCH**, and **PROCEDURE**.

The specificator symbols are: **STRING**, **LABEL**, and **VALUE**.

The logical value symbols are: **TRUE** and **FALSE**.

A special symbol available with the B 5000 is (?). This symbol is printed whenever an illegitimate character code is encountered.

IDENTIFIERS

Identifiers serve as names of important entities of a

program. An identifier is composed of letters and digits and must have a letter as its first character. For example:

W9XBY
ABLE
B47
ENTRYPOINT

There are six types of identifiers: variable, procedure, switch, array, label, and formal parameter.

Variable. A variable identifier is a name given to a single arbitrary quantity. The name allows the quantity to be referenced. The quantity may be changed throughout the program.

Procedure. A procedure identifier is a name given to a closed and self-contained process with a fixed set of arguments; for example, a subroutine. The name allows this procedure body to be referenced.

Switch. A switch identifier is a name given to a program switch. A program switch allows the programmer to make a transfer to one of several statement labels.

Array. An array identifier is a name given to an ordered set of values: the variables of a multidimensional array. Any one or all of these values may be changed throughout the program.

Label. A label is a name given to a particular statement, so that it may be referred to in other statements.

Formal Parameter. A formal parameter is a name given to an argument of a procedure.

NUMBERS

A number is composed of digits, with additional explanatory symbols allowed in order to denote the sign of the number, decimal point, the existence of a power of ten factor, the sign of the power of ten, and the actual power. For example:

−316
+36.001
16 ×10*1
13.62 ×10* −3
−7.5 ×10* −9

STRINGS

A string is a combination of any basic symbols (not containing ") which are bounded by the bracket symbol ". For example:

"/ −1AB;(."
"JAMES"
"1268V896"

EXPRESSIONS

The two levels of language presented thus far, the alphabet and words of ALGOL 60, are used in various combinations to form the next language level, expressions. Before considering the three major expressions—arithmetic, Boolean, and designational—the minor expressions of ALGOL 60, variables and function designators, must be discussed. These minor expressions, along with operators, delimitors and reserved words, constitute the major expressions.

VARIABLES

A variable is a designation given to a single value. This value may be changed at will by means of a particular type of statement in the program. A variable may be either a variable identifier or an array identifier with subscripts. For example:

A[1,2]
B[I,J]
EPSILON
M2[X +Y,J +K]
X

Since vertical alignment of characters has no meaning in this language, subscripts are enclosed by brackets and separated by commas. These subscripted variables designate values which are components of multidimensional arrays; the subscripts designate which value. The subscripts of a list may be variables, function designators, or arithmetic expressions.

FUNCTION DESIGNATORS

Function designators are also used for denoting single values, but with the distinction that these values are the result of a specific set of computations on given parameters. A function designator consists of a procedure identifier, which refers to the procedure body containing the specific set of computations, and an actual parameter part, which contains the list of parameters to be used in the computations. For example:

SIN(X)
LN(2 ×A −3 ×B)
P(X, Y −Z)

The parameters of the actual parameter part are enclosed in parentheses and separated by commas.

The standard functions of analysis and their procedure identifiers are an integral part of the B 5000 programming language; the procedures to which these identifiers refer need not be defined by the programmer in a procedure body. For example:

SIN
ARCTAN
EXP
SQRT
ENTIER

The function ENTIER is used for transferring an expression of real type to one of integer type.

ARITHMETIC EXPRESSIONS

There are two types of arithmetic expressions: simple and general.

Simple. A simple arithmetic expression is a description of an algebraic process which produces a numerical value. It is composed of numbers, variables, function designators, and arithmetic operator symbols. For example:

16.2 × SIN (X + Y)
LN(2 × A − 3 × B)/(−.165*C × EPSILON)
A[1,2] × M2[X + Y, J − K]

When numbers are used for operands, the values of the operands are self-evident. When the operands are variables, the numerical values to be operated on are those currently assigned to the variables by a program statement. When a function designator is involved, the value is derived from the procedure body containing the computations which define the function.

General. A general arithmetic expression also produces a numerical value. It contains several simple arithmetic expressions and a means for selecting the one which is to be evaluated.

Three sequential operator symbols, Boolean expressions (below) and simple arithmetic expressions are combined as follows:

IF (Boolean expression) **THEN** (simple arithmetic expression) **ELSE** (arithmetic expression). For example:

IF X **DIV** Y > 0 **THEN** X + Y **ELSE** Z
IF A < B **THEN** B/A **ELSE IF** A > B **THEN** A/B **ELSE** 1

The selection is based on the values (**TRUE** or **FALSE**) of the Boolean expressions. The Boolean expressions of the **IF** clauses are evaluated in sequence from left to right until one having the value **TRUE** is found. The simple arithmetic expression then selected for the value of the general arithmetic expression is the one following the next **THEN**.

If none of the Boolean expressions is **TRUE**, the simple arithmetic expression evaluated is the last one in the general arithmetic expression; that is, the one following the last **ELSE**.

In the second example listed above, there is a choice of three simple arithmetic expressions: B/A, A/B, and 1. If, in an actual problem, A = 10 and B = 5, the second Boolean expression, A > B would be true and A/B or 2 would be the numerical value of the expression.

BOOLEAN EXPRESSIONS

There are also simple and general Boolean expressions.

Simple. A simple Boolean expression is a description of a process for computing a logical value: **TRUE** or **FALSE**. The expression is composed of logical value symbols, variables (Boolean type), function designators (Boolean type), logical operator symbols and relations (arithmetic expressions connected by a relational operator). For example:

A ∧ B
S ∨ Q **EQV** R
(X − Y) × Z ≤ 5 × P

Whenever variables or function designators are used in Boolean expressions, they must be declared Boolean by a special declaration in the program.

Whatever is located on either side of a logical operator symbol must have a logical value. The elements separated by a relational operator symbol must have a numerical value.

Relational operator symbols have familiar meanings, but logical operators are not so commonly known. Their meaning is given in the following chart:

	IF			
	A	is FALSE	FALSE	TRUE	TRUE
AND	B	is FALSE	TRUE	FALSE	TRUE
	THEN				
	¬B	is TRUE	FALSE	TRUE	FALSE
	A ∧ B	is FALSE	FALSE	FALSE	TRUE
	A ∨ B	is FALSE	TRUE	TRUE	TRUE
	A **IMP** B	is TRUE	TRUE	FALSE	TRUE
	A **EQV** B	is TRUE	FALSE	FALSE	TRUE

General. A general Boolean expression is also a description of a process for computing a logical value. It contains several simple Boolean expressions and a means for selecting the one bearing the value to be used. The selection method is analogous to that used for general arithmetic expressions. A general Boolean expression is formulated as follows: **IF** (Boolean expression) **THEN** (simple Boolean expression) **ELSE** (Boolean expression). For example:

IF L ≥ M **THEN** K = C **ELSE** N < Y
IF JOE **THEN FALSE ELSE IF** PETE **THEN TRUE ELSE** JOE

DESIGNATIONAL EXPRESSIONS

A designational expression is a rule by which a label of some statement is selected for the purpose of referencing that statement. A simple designational expression is either a label or a switch designator.

Labels have been discussed. A switch designator is used to specify the program path to be followed through a switch, which is a conditional transfer device with a number of choices. The switch designator is composed of a switch identifier followed by a subscript expression enclosed in brackets. In order for the switch designator to have meaning, its arithmetic expression part must assume a positive integral value not greater than the number of choices available in the switch. The switch choices are listed in a declaration to which the switch designator refers. The transfer choice selected is the nth label in the declaration switch list counting from left to right, n being the integral value of the arithmetic expression part of the switch designator.

For example:

A75
MABEL
PICK[N +1]

There is also a general designational expression. Its formulation and evaluation principles are entirely analogous to those of arithmetic expressions. For example:

IF A<C **THEN** A75 **ELSE** PICK [N +1]

STATEMENTS

Statements are the programming units of the language; they are comprised of the basic symbols, words, and expressions previously discussed. Statements, like verbal sentences, are complete units of communication. Just as sentences can be combined to form paragraphs, so can basic statements be combined to form compound statements.

For referencing purposes, any basic statement can be given a label. So can a compound statement or a block be labeled. A compound statement might appear as follows: label: **BEGIN** statement; statement; statement;...statement; statement **END**. A block might appear as follows: label: **BEGIN** declaration; declaration;...declaration; statement; statement;... statement; statement **END**

Every block automatically introduces a new level of nomenclature; that is, an identifier occurring within a block may, through a declaration, be specified to be local to the block.

Declarations serve to define certain properties of the identifiers of the program. For example, a procedure declaration defines the procedure associated with a procedure identifier. Since it is necessary to refer to declarations during the descriptions of the statement types, the brief description of them given above was imperative at this point.

There are six types of statements possible in an ALGOL 60 program.

ASSIGNMENT STATEMENTS

An assignment statement serves to give a specific value to one or several variables. Each entry in the statement is separated by the symbol ←, with the variables listed to the left and the expression by which they are to be replaced at the right. For example:

P ← P +1
M2[X +Y,Z]←A [1,2] ← SIN(C −D)
OUT ← **TRUE**
IN ← Q ∧ R

As used in assignment statements the symbol ← means "Using the current value of the variables, evaluate the expression. The result is then assigned to all of the variables to the left of the symbol (←)". All variables on the left must be of the same type. If the variables are Boolean, the expression must likewise be Boolean. If the variables are of type real or integer, the expression must be arithmetic.

GO TO STATEMENTS

A go to statement comprises the sequential operator **GO TO** and a designational expression. For example:

GO TO MABEL
GO TO PICK [N +1]
GO TO IF A<C **THEN** A75 **ELSE** PICK [N +1]

A go to statement is used for transfer of control. It serves to interrupt the normal sequence of operations from one statement to the next. The label of the next statement to be executed after a go to statement is defined by the value of the go to statement designational expression.

DUMMY STATEMENTS

A dummy statement executes no operation. For example:

BACK:
BEGIN...; JOHN: **END**

The primary function of a dummy statement is to place a label at the end of a procedure so that transfer of control can be made midway through the procedure back to the place in the program which called for its execution.

CONDITIONAL STATEMENTS

A conditional statement can cause certain statements to be executed or skipped depending on the values of specified Boolean expressions. A conditional statement in general takes the following form: **IF** Boolean expression **THEN** unconditional compound statement. The **ELSE** portion is arbitrary.
For example:

IF A>B **THEN** X←X+1

IF X=0 **THEN** X←1 **ELSE IF** X=9 **THEN**
GO TO BACK

IF E+F=6 **THEN** G←1 **ELSE GO TO** OUT

If the Boolean expression of the **IF** clause is **FALSE**, the statement following it will be skipped and operation will continue to the next statement encountered, either the statement following the sequential operator **ELSE** or the next statement in sequence. If the Boolean expression of the **IF** clause is **TRUE**, the statement following it is executed and operation then continues with the next statement in sequence beyond the conditional statement.

FOR STATEMENTS

A **FOR** statement provides the ability to perform repetitive operations on a statement which is a part of it. The basic structure of a **FOR** statement is as follows: **FOR** variable ← list **DO** compound statement.

A **FOR** statement causes the statement following the **DO** to be repeatedly executed zero or more times with a new assignment to its controlled variable each time. The list provides a rule for obtaining the values which are assigned. The sequence of values is obtained from the list elements (arithmetic expression, **STEP-UNTIL**, or **WHILE**) by taking these one by one in the order in which they are written.

Arithmetic Expression Element. The simple case is a single arithmetic expression, the value of which is calculated immediately before the single execution of the statement.

Step-Until Element. This element takes the form A **STEP** B **UNTIL** C, where A, B, and C are arithmetic expressions. This element causes the assignment of the current values of A, A+B, A+2×B, etc., to the variable and the corresponding execution of the statement following **DO**. The operation terminates (list exhausted) when V (variable) − C is nonzero and has the same sign as B. This test is made just prior to each execution of the statement following **DO**, so that the statement is never executed if initially A − C and B have equal signs, if both are nonzero.

While Element. This element takes the form E **WHILE** F, where E is an arithmetic and F a Boolean expression. Each time the variable is assigned the value of the arithmetic expression, a test is made on the Boolean expression. If it has a value of **FALSE**, the list is exhausted. If it is **TRUE**, the statement following **DO** is executed.

PROCEDURE STATEMENTS

A procedure statement calls for the execution of a procedure declaration body. It is composed of a procedure identifier and an actual parameter list enclosed in parentheses and separated by commas. For example:

TRANSPOSE (W,V+1)
ABSMAX (A,N,M,Y,I,K)

The number of actual parameters listed must agree with the number of formal parameters given in the procedure declaration heading.

DECLARATIONS

Declarations define certain properties of identifiers of a program. A declaration for an identifier is valid for one block. Outside this block the particular identifier may be used for other purposes.

A declaration may be marked with the declarator symbol **OWN**. Upon re-entry into a block, values of **OWN** quantities will be unchanged from their values at the last exit; others are undefined.

TYPE DECLARATIONS

Values of simple variables may be of three types: **INTEGER**, **REAL** or **BOOLEAN**. Type declarations place simple variables into one of these three classes. A type declaration is composed of a declarator symbol followed by all simple variables of that class. The simple variables are separated by commas. For example:

REAL W9XBY, X, EPSILON
INTEGER ABLE, B47, ENTRYPOINT
BOOLEAN Y, Z, BAKER

Simple variables declared as **REAL** may assume positive and negative values including zero. Those declared as **INTEGER** may assume only positive and negative integral values including zero.

The constituents of a simple arithmetic expression may be either of type **REAL** or type **INTEGER**. In general, if there is a mixture of types within a simple arithmetic expression, the resulting value will be **REAL**. If all constituents are **INTEGER**, the resulting value is **INTEGER**.

ARRAY DECLARATIONS

An **ARRAY** declaration specifies one or more identifiers to represent multidimensional arrays and gives the array dimensions, the upper and lower bounds of the subscripts, and the types of the variables. An **ARRAY** declaration comprises declarator symbols, array identifiers, and bound pair lists. For example:

INTEGER ARRAY A[1:6, 1:3], B[−N:O, O:N]
ARRAY M2 [**IF** X<0 **THEN** 0 **ELSE** 1:6],
MA[C:D]

A bound pair is two arithmetic expressions separated by the symbol (:). The first expression is the lower bound and the second the upper bound of a subscript. A bound pair list is a series of bound pairs separated by commas and giving the bounds of all subscripts of an array taken in order from left to right. The bound pair list immediately follows the identifier to which it pertains.

The dimension of an array is given as the number of entries in the bound pair list.

All arrays specified in one declaration are of the same quoted type. If no type is quoted in an **ARRAY** declaration, type **REAL** is understood.

SWITCH DECLARATIONS

A switch declaration lists all program transfer choices available at any one switch. It is composed of the symbol **SWITCH**, a switch identifier, the symbol (←) and a list of designational expressions separated by commas. For example:

SWITCH PICK←P1, P2, P1 + 6
SWITCH Y←A75, MABEL, **IF** A<C **THEN**
A75 **ELSE** PICK[2]

A **SWITCH** declaration is referred to each time a switch designator is encountered in the program. The transfer choice taken is the nth expression of the **SWITCH** list, where n is the value of the arithmetic expression associated with the switch designator. The expression selected is then evaluated using the current values of all variables involved.

PROCEDURE DECLARATIONS

A procedure declaration defines in detail the procedure associated with a procedure identifier. A procedure declaration is made up of a heading and a body.

Procedure Declaration Heading. The heading begins with the symbol **PROCEDURE** followed by the procedure identifier. Next comes a formal parameter part which is a list of formal parameters enclosed in parentheses and separated by commas. The formal parameter part is followed by a value part. The value part starts with the symbol **VALUE** and is followed by the formal parameters which are to be replaced by the values of the actual parameters when the procedure body is executed. The last part of the heading is a specification part which describes each formal parameter. It may specify that it is an **ARRAY**, **STRING**, **PROCEDURE**, **LABEL**, **SWITCH**, etc. Specificator symbols and/or declarator symbols are given followed by the formal parameters which are of that type. A sample procedure declaration heading follows:

PROCEDURE POLYROOT (A,X,Q1, TRIG);
VALUE A, X; **REAL** A, X; **ARRAY** Q1;
LABEL TRIG

Procedure Declaration Body. The procedure declaration body comprises a statement or piece of code. The body may be activated from other parts of the block in the head of which the procedure declaration appears by means of procedure statements and/or function designators.

Before execution of the procedure body takes place (this is caused by execution of a procedure statement), all formal parameters quoted in the value part of the procedure declaration are replaced by their corresponding actual parameter values. Formal parameters not quoted in the value list are replaced, throughout the procedure body, by the actual parameters themselves, not the values thereof.

Finally the modified procedure body is inserted in the place of the procedure statement in the program sequence and executed.

ALGOL EXAMPLE

To give the reader a feeling for this language an example follows. The problem is to find the Gamma Function.

$Y = \Gamma(z)$

The method chosen for the calculation is based on the fact that $\Gamma(1+x)$ can be approximated by an eight degree polynominal of the form:

$$F(x+1) = 1 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 + a_5x^5 + a_6x^6 + a_7x^7 + a_8x^8$$

where:

$a_1 = -.57719165$	$a_5 = -.75670408$
$a_2 = .98820589$	$a_6 = .48219939$
$a_3 = -.89705694$	$a_7 = -.19352782$
$a_4 = .91820686$	$a_8 = .035868343$

The job is to write a subroutine which can be used by any program which requires the evaluation of a Gamma Function. The range of the variable for

which this approximation of the Gamma Function is defined is:

$0<z<41.15$

It is assumed here that any program making use of this routine will, before calling for its execution, have determined that the variable is within the defined range.

One of three different formulas applies, depending upon the value of the variable within the defined range. They are:

1. If $0<z\leq 1$ $\quad Y = \frac{1}{z} F(1+z)$ is used.
2. If $1<z\leq 2$ $\quad Y = \Gamma[1+(z-1)]$ is used.
3. If $z>2$ $\quad Y = (z-n)(z-n+1)...(z+1)$ $F(z-n)$ is used. where $1\leq z-n<2$

For $z>2$ the integer n allows for the introduction of a new variable, $z-n$, which is then in the second range above. The function $\Gamma(z-n)$ can then be calculated by means of the second formula. The third formula gives the relationship between $\Gamma(z)$ and $\Gamma(z-n)$.

After making the necessary substitutions, the above three equations become:

1. $Y = \frac{1}{z}(1 + a_1z + a_2z^2 + a_3z^3 + a_4z^4 + a_5z^5 + a_6z^6 + a_7a^7 + a_8z^8)$
2. $Y = 1 + a_1(z-1) + a_2(z-1)^2 + a_3(z-1)^3 + a_4(z-1)^4 + a_5(z-1)^5 + a_6(z-1)^6 + a_7(z-1)^7 + a_8(z-1)^8$
3. $Y = (z-n)(z-n+1)...(z+1) \times [1 + a_1(z-n-1) + a_2(z-n-1)^2 + a_3(z-n-1)^3 + a_4(z-n-1)^4 + a_5(z-n-1)^5 + a_6(z-n-1)^6 + a_7(z-n-1)^7 + a_8(z-n-1)^8]$

where $n = \text{ENTIER}(z-1)$

ENTIER is a function which produces an integral value from an expression which may be composed of nonintegral values. For instance, n = ENTIER (z −1) means that if z = 4.5, then n = 3 or the largest integer which is not greater than the function argument (z − 1.)

It can be noted here that ENTIER is a standard function in the B 5000 language and is automatically available for use by the programmer, as are others, such as SIN, ABS, SQRT, ARCTAN, LN, etc.

A solution of this problem using the ALGOL part of the B 5000 programming language is given below. The keypunch operator would punch exactly that which is shown.

```
PROCEDURE GAMMA (Z,Y); VALUE Z;
REAL Z,Y; INTEGER M, N;
  BEGIN
    REAL PROCEDURE POLYGAM (X);
     VALUE X; REAL X;
      BEGIN
        REAL Y;
      Y←1 − .57719165 × X + .98820589
        × X*2 − .89705694 × X*3 + .91820686
        × X*4 − .75670408 × X*5
        + .48219939 × X*6 − .19352782
        × X*7 + .035868343 × X*8;
       POLYGAM←Y
      END

    COMMENT: START OF GAMMA PRO-
     CEDURE PROGRAM;
    IF Z<1 THEN Y ← POLYGAM (Z)/Z
      ELSE IF Z≤2 THEN Y←POLYGAM
       (Z −1)
        ELSE BEGIN
        M←ENTIER (Z −1); Y←POLYGAM
         (Z −M −1);
        FOR N←M STEP −1 UNTIL −1
         DO Y←Y × (Z −N)
        END
  END
```

NOTE

Even in a problem-oriented language the programmer can make use of his ingenuity by formulating expressions in ways that decrease the running time of object programs. For example, the first assignment statement of the procedure (POLYGAM) could have been written as Y← (((((((.035868343 × X − .19352782) × X + .48219939) × X − .75670408) × X + .91820686) × X − .89705694) × X + .98820589) × X − .57719165) × X + 1

A solution to the same problem written in the machine language of another computer has been formulated as follows. Again, only what is to be punched by the keypunch operator is shown.

00000047002	00000000000
00000080020	05110000000
00000737002	00000807048
00000127019	00000127049
20000020038	20000020069
00000647001	00000647047
00000140004	00000837049
20000020056	20000020059
00000110000	20000300050
20000020057	05110000000
00000647019	05110000000
00000817018	00000000000
00000737012	00000727058
20000380040	00000827069
00000757018	10000807060
20000300020	00000227051
00000000000	00000827059
00000000000	00000727057
05110000000	00000000000
00000000000	00000000000
00000737020	00000000008
20000380070	00000000000
00000647038	05110000000
00000817036	15057719165
00000127038	05098820589
00000827037	15089705694
00000127037	05091820686
00000647038	15075670408
00000807035	05048219939
00000737035	15019352782
00000287022	04935868343
00000807039	00000000000
20000020069	00000747075
00000647037	20000020069
20000300073	00000647075
15120000000	20000020059
05110000000	20000300050
05110000000	05110000000

Even though it is a challenging and to some an interesting exercise to produce an actual machine language program, it seems obvious from the above examples that if one were interested in getting an answer as fast as possible, he would choose the former method.

In order to establish an association between the syntactic rules of ALGOL and the ways in which those rules are used in writing a program, the previous solution is presented (Pg.9) with individual parts identified. All English words which serve as basic symbols of the language are printed boldface.

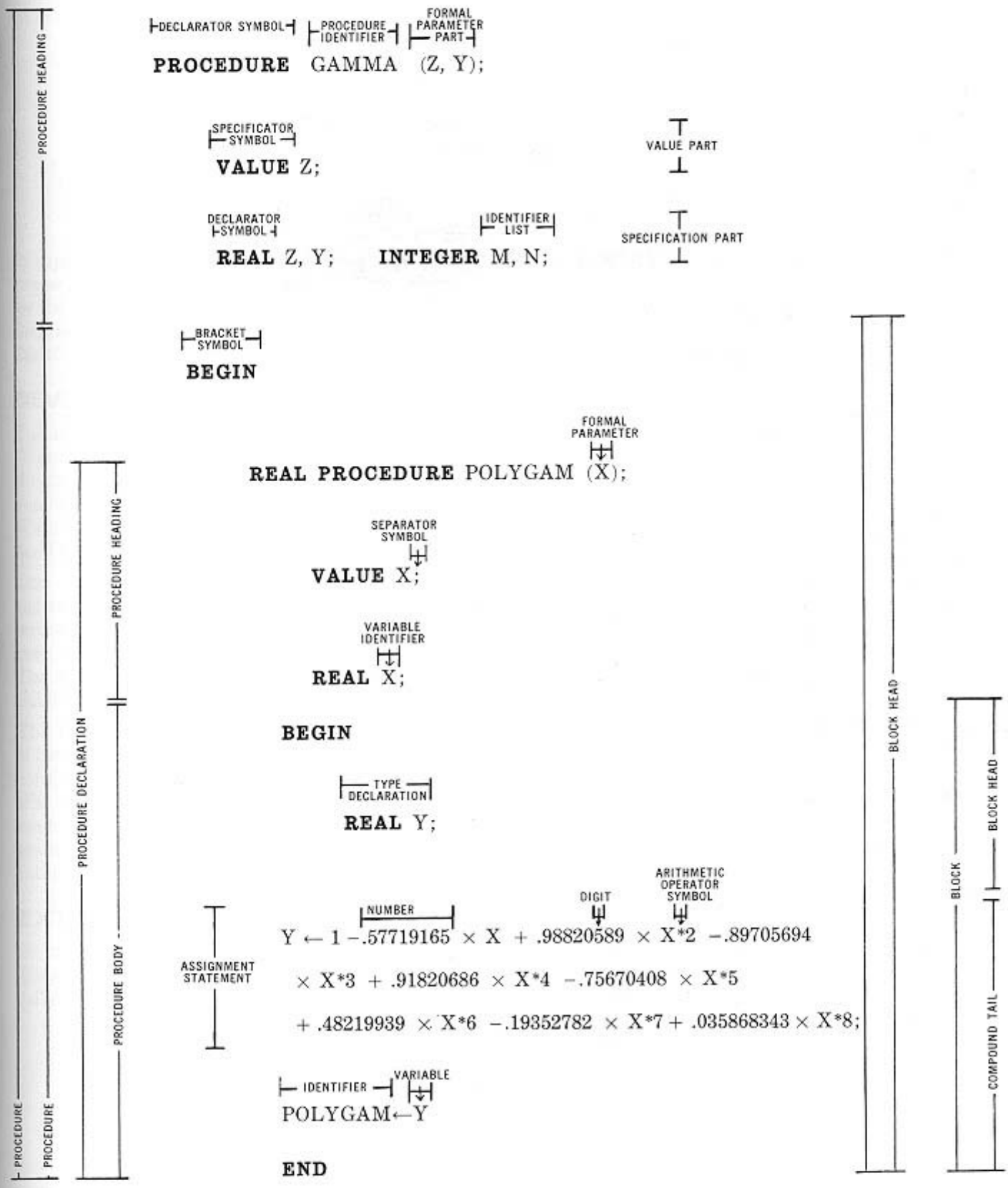
The question may arise as to how this procedure declaration is fitted into another program. Examination of the construction of an over-all program yields the answer.

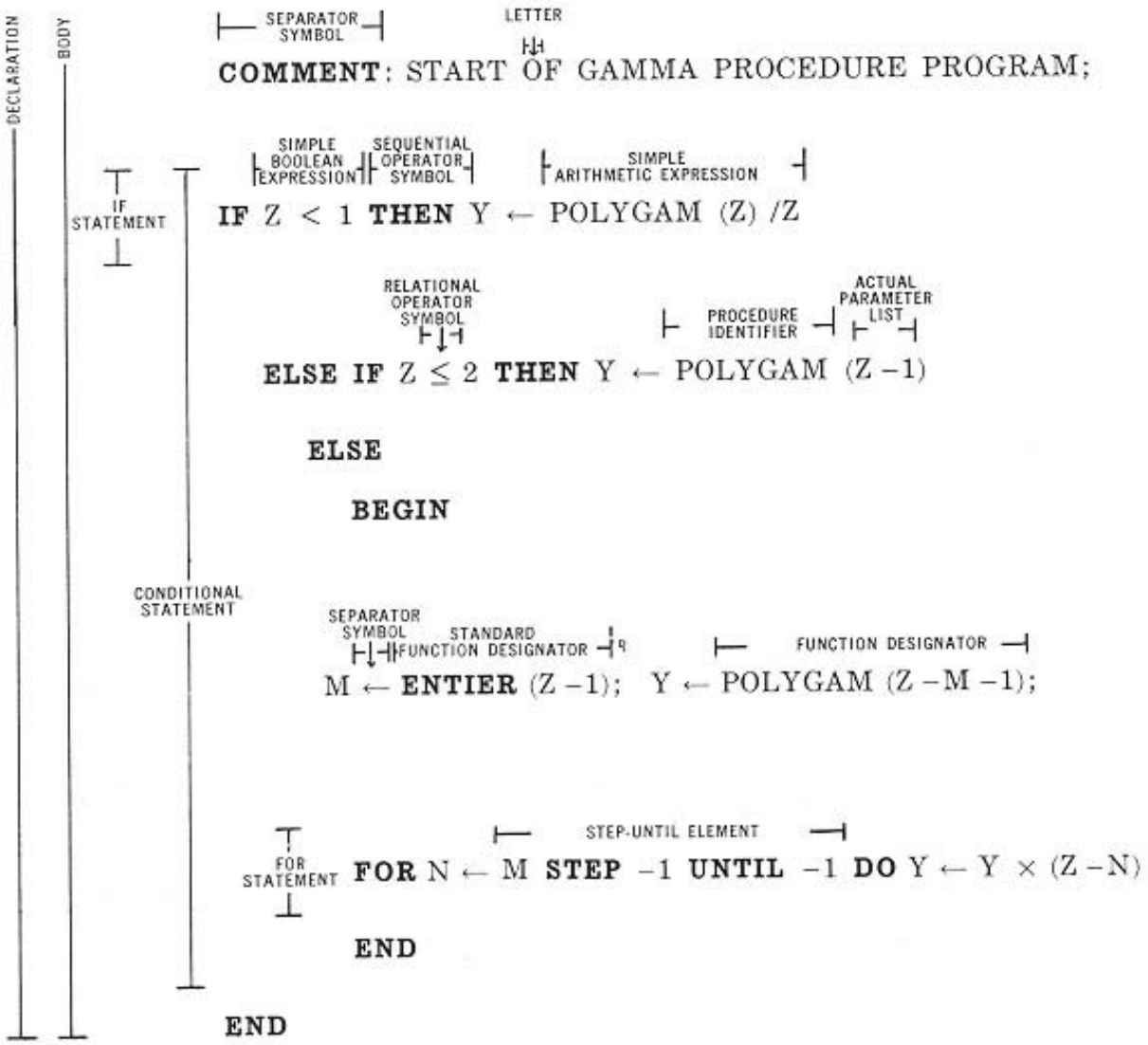
Sections of a program are called blocks or compound statements. Both are bounded by the bracket symbols **BEGIN** and **END**. It is interesting to note that the creators of ALGOL could have chosen single characters for this purpose such as (&). Instead, English words were chosen because of their familiar meaning. The only difference between a block and a compound statement is that a block contains declarations immediately after the left bracket symbol, **BEGIN**, and compound statements do not. It can be seen then that the declaration outlined above for evaluating a Gamma Function must appear in the head of some block of the program that uses it.

In order to initiate its use, the program must also contain a procedure statement of the form:

	Procedure Identifier	Actual Parameters
Procedure Statement	GAMMA	(BETA, MU)

Writing the above statement in the program would result in MU assuming the value of Γ (Beta).


DECLARATOR SYMBOL
PROCEDURE IDENTIFIER
FORMAL PARAMETER PART
PROCEDURE GAMMA (Z, Y);
SPECIFICATOR SYMBOL
VALUE Z;
VALUE PART
DECLARATOR SYMBOL
IDENTIFIER LIST
REAL Z, Y; INTEGER M, N;
SPECIFICATION PART
BRACKET SYMBOL
BEGIN
FORMAL PARAMETER
REAL PROCEDURE POLYGAM (X);
SEPARATOR SYMBOL
VALUE X;
VARIABLE IDENTIFIER
REAL X;
BEGIN
TYPE DECLARATION
REAL Y;
NUMBER
DIGIT
ARITHMETIC OPERATOR SYMBOL
Y ← 1 −.57719165 × X + .98820589 × X*2 −.89705694
× X*3 + .91820686 × X*4 −.75670408 × X*5
+ .48219939 × X*6 −.19352782 × X*7 + .035868343 × X*8;
ASSIGNMENT STATEMENT
IDENTIFIER
VARIABLE
POLYGAM←Y
END
PROCEDURE HEADING
PROCEDURE DECLARATION
PROCEDURE BODY
PROCEDURE
PROCEDURE
BLOCK HEAD
BLOCK
BLOCK HEAD
COMPOUND TAIL


DECLARATION
BODY
SEPARATOR SYMBOL
LETTER
COMMENT: START OF GAMMA PROCEDURE PROGRAM;
IF STATEMENT
SIMPLE BOOLEAN EXPRESSION
SEQUENTIAL OPERATOR SYMBOL
SIMPLE ARITHMETIC EXPRESSION
IF Z < 1 THEN Y ← POLYGAM (Z) /Z
RELATIONAL OPERATOR SYMBOL
PROCEDURE IDENTIFIER
ACTUAL PARAMETER LIST
ELSE IF Z ≤ 2 THEN Y ← POLYGAM (Z −1)
ELSE
BEGIN
CONDITIONAL STATEMENT
SEPARATOR SYMBOL
STANDARD FUNCTION DESIGNATOR
FUNCTION DESIGNATOR
M ← ENTIER (Z −1); Y ← POLYGAM (Z −M −1);
FOR STATEMENT
STEP-UNTIL ELEMENT
FOR N ← M STEP −1 UNTIL −1 DO Y ← Y × (Z −N)
END
END

APPENDIX C

GLOSSARY

(Some terms defined are characteristic to the industry, but others are significant only with respect to the B 5000 System. For a more complete glossary of industry terminology it is recommended that reference be made to glossaries published in such trade journals as ACM Communications and Computers and Automation.)

Address—A label, such as an integer or other set of characters, which identifies a memory location or storage device.

ALGOL—(for ALGOrithmic Language) an international problem language designed for the concise, efficient expression of arithmetic and logical processes, and the control (iterative, etc.) of these processes.

Algorithm—A statement of the steps to be followed in the solution of a problem.

Argument—Known reference factor necessary to find the desired item in a table or array. Sometimes referred to as a "key" as in "search key."

Array—An ordered arrangement of items of information.

Automatic Programming—Technique which employs the computer itself to translate programming from a form that is easy for a human being to produce and understand into a form suitable for use by a computer.

Binary—A radix-2 number system using only the digits 0 and 1.

Boolean Algebra—A system of algebra dealing with truth values as variables and having basic operators such as "and," "or," "not," etc.

Boolean Variables—An operand in a Boolean algebra expression. A Boolean variable may have the value of "true" or "false," commonly represented in computers by one and zero respectively.

BURROUGHS Common Language (BCL)—A binary code representation of alphanumeric characters common to all future BURROUGHS equipment and common with existing standard punched-card and tape representations.

Call—A set of characters or bits which demand an action to take place or some item of information; for example, subroutine call, operand call, descriptor call.

Channel—A path along which information may flow. (See Input/Output Channel.)

Characteristic—The exponent portion of a floating-point number. (See Floating-Point Representation.)

Clock—A time-increment counting register used for program-interrupt and job-time accounting.

COBOL—A COmmon Business Oriented Language designed for expressing problems of data manipulation and processing in English narrative form.

Compiler—A translator program which reduces a problem-oriented language into the machine language of a particular computer.

Concatenating—Linking together by forming a chain or series, a series or order of things depending on each other.

Control Counter—A 17-bit register which indicates the location of the next syllable to be executed by the Processor.

Data Array—Any ordered set of data, such as the information on a card, a tape record, a print line, the contents of a working area, etc.

Data-Manipulation Mode—One of the logical operational modes of the B 5000, in which the basic information unit is a single alphanumeric character. Utilized for efficient editing, formating, and comparison functions.

Data Manipulators—A set of operators which edit, compare, and move data within memory when the Processor is in the Data-Manipulation mode.

Debug—To isolate and correct the mistakes in a program.

Descriptor—A computer word used specifically to define characteristics of a program element. For example, descriptors are used for describing a data record, a segment of a program, or an input-output operation.

Descriptor Call Syllable—A syllable of the B 5000 program string which directs the Processor to place in the Stack the location of a data array or a program segment.

Diagnostic Routine—Routine designed to detect and locate either a malfunction of the system or a mistake in programming.

Drum, Magnetic—A rapidly rotating cylinder, the

surface of which is coated with a magnetic material on which information may be stored as small magnetized areas.

Edit—The act of arranging information from input-output devices. This may involve the selection of pertinent data, the insertion of symbols such as page numbers and check-protection characters, and standard processes such as zero suppression.

Executive Routine—A routine designed to control and cause the execution of other routines. (See Master Control Program.)

Field—A set of one or more characters which is treated as a unit of information.

Fixed-Point Representation—An arithmetic notation in which all numeric quantities are expressed by the same number of digits with the decimal point (for base 10) or octal point (for base 8) assumed in a fixed location in each number. Alignment of numbers with different assumed locations of the points must be performed by the program before an arithmetic operation such as addition can be performed.

Floating-Point Representation—An arithmetic notation in which all numeric quantities have an associated indication of the decimal point location (base 10) or octal point location (base 8). Automatic alignment of numbers and calculation of the location of the point can be provided in arithmetic on floating-point numbers. In the B 5000, a floating-point number consists of two parts: a 13-digit octal integer with sign called the mantissa; and a signed number called the characteristic (or exponent) which indicates the number of places to the right or left that the actual octal point is from the assumed octal point in the mantissa.

Hardware—The mechanical, magnetic, electrical, and electronic devices from which a computer system is constructed.

Housekeeping—Operations not directly concerned with the objective of a program; e.g., packing or rearranging data, subroutine linkages, etc.

Indirect Address—An address which identifies a memory cell containing an address. The contents of the memory cell is the address of the desired information or may also be an indirect address.

Input/Output Channel—A device which allows independent, simultaneous communication between any Memory Module and any of the several input-output units. It controls any peripheral device and performs all validity checking on information transfers.

Input/Output Exchange—An electronic switch which connects an Input/Output Channel to the designated peripheral device.

Interrupt—A signal generated by an input-output device, by an operational error, or by a request by the Processor for more data or program segments. Provides the Master Control Program with the facility to maintain control of all system functions.

Jump—An operation which may alter the normal sequence of a program. Normally syllables are executed in sequence; a jump operation causes a termination of the sequence and directs the Processor to a specified syllable. A conditional jump operation is a jump operation which takes place only if a specific condition exists in the Processor. Usually the condition is a result of a test or comparison operation. If the specific condition does not exist, a conditional jump operator is ignored and sequential execution of syllables continues.

Library—Collection of fully tested standard programs and subroutines for repeated use by, or incorporation into, other programs.

Literal—An element in a program which is itself a quantity or alphanumeric constant to be used by the program rather than being an address of the quantity or constant.

Machine Language—The coded operations that control information and addresses employed within the Processor to express a program. (See Problem Language.)

Mantissa—Integer part of a floating-point number (13 octal digits in a single-precision number of the B 5000). (See Floating-Point Representation.)

Master Control Program—A computer program to control the operation of the system. It is designed to reduce the amount of intervention required of the human operator. The Master Control Program performs the following functions: schedules programs to be processed; initiates segments of programs; controls all input-output operations to insure efficient utilization of each system component; allocates memory dynamically; issues instructions to the human operator and verifies that his actions were correct; performs corrective action on errors in a program or system malfunction.

Megacycle/Sec.—A million cycles per second. The basic pulse rate of the B 5000 is 1 megacycle/second.

Memory—Internal computer storage. Distinguished from other types of storage in the B 5000 which are part of the peripheral equipment.

Memory Exchange—An electronic switching device which controls information flow among Memory

Modules and the Processor or Input/Output Channels.

Microsecond—One millionth of a second (0.000001 sec. or 1 μs).

Modularity—The property of a system resulting from the construction or assembly of the system from logical subunits (modules). In the B 5000, this property provides the capability of constructing a system with the proper number of each type of module to match varying processing requirements efficiently and to maximize the utilization of each module.

Module—A logical subunit that may be easily detached from, or included with, the whole system. Processor, Magnetic Tape Units, and Storage Drums are typical modules of the B 5000 System.

Multi-Processing—Processing several programs or program segments concurrently on a "time-share" basis. The Processor is only active on one program at any one time while operations such as input-output may be performed in parallel on several programs. The Processor is directed to switch back and forth among programs under the control of the Master Control Program.

Nesting—Enclosing one program element of a particular type, such as a subroutine, within another of the same type.

Noisy Mode—A mode of floating-type arithmetic operation in which the error resulting from use of only a finite number of significant digits may be identified.

Normal Mode—The standard B 5000 operational logic used during computational processes. The basic information unit is the word.

Object Program—A set of machine-language instructions for the solution of a specified problem, obtained as the end result of the compilation process (see Compiler, Problem Language).

Octal—A number system based on powers of 8 rather than 10 as in the decimal system. Includes only the digits 0, 1, 2, 3, 4, 5, 6, and 7.

Operand—Any of the quantities entering into an operation. An operand is typically a number for arithmetic operations. For comparison operations, an operand may be an alphanumeric field.

Operand-Call Syllable—A syllable which specifies that an operand be brought to the Stack, either directly from the Program Reference Table or indirectly by means of a descriptor.

Operators—Symbols that denote a fixed, predefined set of operations to be performed in a specified sequence. There are a number of classes of operators in the B 5000: for example, the arithmetic operators are $+$, $-$, $\times$, $/$, DIV; the relational operators are $<$, $\leq$, $=$, $>$, $\geq$, $\neq$.

Output Channel—(See Input/Output Channel.)

Parallel Operation—Flow of data through the system or any part of it, using two or more communication lines or channels simultaneously.

Parallel Plate Packages—A packaging technique for logical (and other) computer circuitry developed by BURROUGHS to achieve high packing density, ease of automatic production and assembly, and simple maintenance.

Parallel Processing—Processing more than one program at a time on a parallel basis, where more than one Processor is active at a time (distinguished from Multi-Processing where only one Processor is active on one program at a time).

Parameter—In a subroutine, a quantity which may be given different values when the subroutine is used in different parts of one main routine but which usually remains unchanged throughout any one such use. To use a subroutine successfully in many different programs requires that the subroutine be adaptable by changing its parameters.

Parity Check—A summation check in which the binary digits, in a character or word, are added (modulo 2) and the sum checked against a single, previously computed parity digit.

Peripheral Equipment—Any of the several devices, primarily used to communicate with a system, not considered a part of the main processing and control system. On the B 5000, the peripheral equipment includes Magnetic Tape Units, Line Printers, Card Readers, Card Punches, Keyboard, Message Printer, and Plotter.

Polish Notation—A method of writing logical and arithmetic expressions without the need for parentheses, originated by the Polish logician J. Lukasiewicz. For example: Normal algebraic notation $(X+Y) \times (A-B)$, in Polish notation: $XY+AB-\times$.

Precision—The degree of exactness with which a quantity is stated. For example, the number 2.783 is precise to four digits, but does not necessarily have four digits of accuracy.

Presence Bit—A single flag bit appearing in descriptors to indicate whether or not the information to which reference is made by the descriptor is in high-speed (core) memory at this time.

Priority—A value assigned to a program or program segment to specify the relative processing se-

quence. The priorities of all programs to be run are taken into consideration by the Master Control Program in arriving at a schedule.

Program Independent Modularity—Property of the B 5000 to accept changes in system configuration and adjust programs accordingly to yield maximum utilization of all modules without reprogramming or recompilation of programs.

Problem Language—The language used by the programmer to state the definition of a problem. ALGOL and COBOL are examples of problem languages. Problem languages are closely related to the type of problem being stated—i.e., algebraic statements for mathematical problems (ALGOL) and narrative English statements for commercial problems (COBOL). Problem language should not be confused with machine language. A program is written in problem language by the programmer. This source program is then translated to the object program (in machine language) by a compiler program. (See Object Program, Compiler.)

Program (noun)—A plan for the solution of a problem. A B 5000 program may be a statement of the problem in ALGOL or COBOL or the translated, segmented object (compilation result) program.

Program (verb)—To plan a computation or process from the original statement of the problem to the delivery of the results, including the integration of the operation of the resulting program into an existing system (for conventional computers). For the B 5000: A system analysis and statement of the problem in common language.

Programming System—The B 5000 Programming System consists of the ALGOL and COBOL compiler and the Master Control Program. The ALGOL and COBOL compiler is loaded by the Master Control Program.

Program Reference Table (PRT)—An area in memory for the storage of operands, references to operands, references to segments of a program, and other program variables. Permits programs to be independent of the actual memory locations occupied by data and parts of the program. Thus programs and data can be placed into any available memory areas without modification to the program.

Real Variable—A variable over the rational and irrational classes of numbers. In ALGOL a real variable is a floating-point number as distinct from an integer variable which is an integer.

Register—The hardware for storing one or more computer words or for maintaining internal system control.

Relocatability—A facility whereby programs or data may be located any place in memory at different times without requiring modification to the program. In the B 5000, segments of the program and all data are independently relocatable with no loss in efficiency.

Return—An operator in a subroutine which recovers all pertinent information from the Stack and transfers control to the next syllable in the original syllable string of the program which causes entry to the subroutine.

Return Point—The syllable in the program segment to which control is transferred after the completion of a subroutine or an intercession by the Master Control Program.

Scheduling—Designation of times and sequence of projected operations. One of the functions of the B 5000 Master Control Program.

Segment (verb)—To divide a program into an integral number of parts, each of which performs some part of the total program and is capable of being completely stored in internal memory.

Simultaneity—Concurrent communication between various units of a system at the same instant.

Software—Programs, routines, and procedures which augment and support a computer system (the Master Control Program, compilers, etc.).

Stack—A portion of memory and /or registers used for temporarily holding information. A Stack, as used in the B 5000, operates on the "last-in first-out" principle, that is, the last item of information placed in the Stack will be the first item of information used when information is required from the Stack. Operators perform their operations on information at the top of the Stack. (See Operators.)

Storage—Any device into which information can be copied, which holds this information, and from which the information can be obtained at a later time.

Storage Allocation—Assignment of specific memory addresses to individual program elements (done automatically in the B 5000 at object running time by the Master Control Program).

Subroutine—The set of instructions necessary to carry out a defined operation; a subunit of a program.

Subroutine Call—A set of characters or lists which initiate a subroutine and contain the parameters or identification of the parameters required by

the subroutine.

Syllable—A literal value, a data-manipulation instruction, a descriptor-call instruction, or an operand-call instruction—the basic unit of B 5000 program strings. Each syllable requires one-fourth of a computer word.

Syntax—Connected system or order of symbol arrangement, the rules or grammar of a language.

Through-Put—The total productive work of a system. Taken into account in determining Through-Put are compilation, debugging, and production times on the system and the system time required by the operators in starting and stopping the jobs. In a broader sense, Through-Put applies to the productive work accomplished by the combination of programmers, system operators, and the system.

Time-Sharing—Interruption of the operation of the main program in a computer by subsidiary or unrelated calculations, with the object of making economic use of computer speed when this is disproportionate to input-output speeds. Also, time-sharing refers to the ability of a module to refer to memory when the Memory Module is not being referenced by some other module. Since some modules make infrequent references to memory during their operation, by interleaving the references several modules can appear to be having access to a Memory Module simultaneously.

Translate—To produce a statement in one language equivalent in meaning to a statement in a different language.

Word—A set of characters or binary digits which occupies one storage location and is treated by the computer as a unit and transferred as such. A word in the B 5000 may contain 8 alphanumeric characters, an octal value in common fixed-floating-point notation, four instructions or literal syllables, or a program or data descriptor.