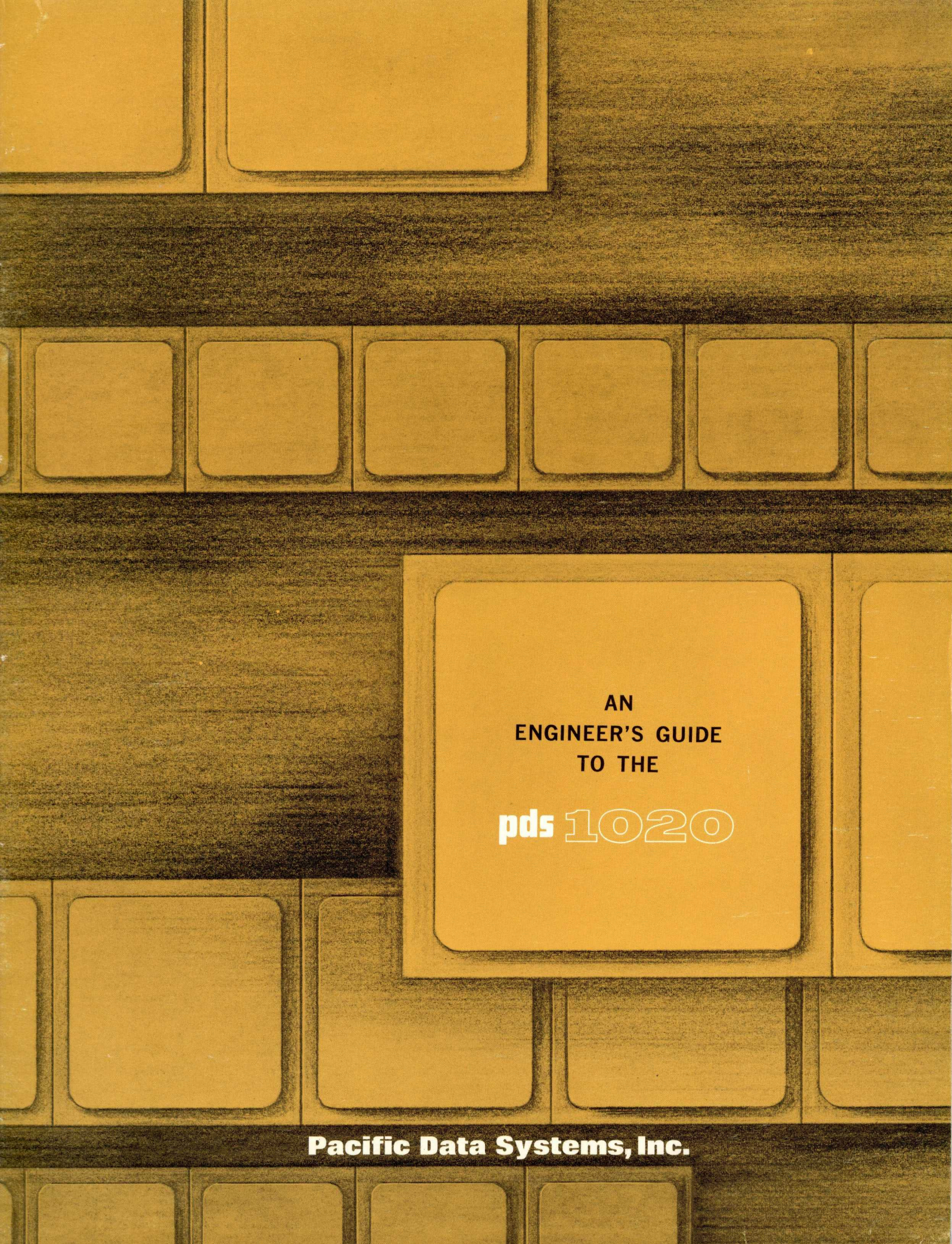

AN
ENGINEER'S GUIDE
TO THE
pds 1020
Pacific Data Systems, Inc.

AN ENGINEER'S GUIDE

To The

PDS 1020 COMPUTER

PACIFIC DATA SYSTEMS
Publications Dept.
April, 1964

PACIFIC DATA SYSTEMS, INC.
A Subsidiary of Electronic Associates, Inc.
1058 East First Street
Santa Ana, California

EG 2-3M

TABLE OF CONTENTS

1.	INTRODUCTION	Page 5
2.	GENERAL DESCRIPTION	" 11
	Use of the Engineering Interpreter	" 11
	Description of the PDS 1020	" 11
3.	STEP-BY-STEP PROCEDURE	" 13
	Input	" 13
	Compute	" 14
	Output	" 18
4.	RETAINED OPERATIONS	" 21
	General Format of RETAIN Command	" 21
	Entering Data	" 21
	Comparison to Step-by-Step Procedure	" 23
	Storage Capacity	" 23
5.	TESTING, JUMPING AND LOOPING	" 24
	Transfers	" 24
	Unconditional	" 24
	Conditional	" 24
	Sub-Programs	" 25
	Example 2	" 26
	Loops	" 27
	Example 3	" 27
	Example 4	" 28
	Address Modification	" 29
	Example 5	" 29
6.	PAPER TAPE EQUIPMENT	" 31
	Paper Tape Reader	" 31
	Paper Tape Punch	" 31
7.	MACHINE OPERATING PROCEDURE	" 33
	Starting the Computer	" 33
	Switch Settings	" 33
	Trouble-Shooting	" 33
	Loading the Paper Tape Reader	" 36
	Loading the Paper Tape Punch	" 37
8.	AND WHERE DO WE GO FROM HERE?	" 38
	What Is An Interpreter?	" 38
	Interpreter Advantages	" 38
	Added Capabilities	" 39
	What Next?	" 39
	APPENDIX	" 41

NOTE TO THE READER

The PDS 1020 Computer is a problem-solving tool, designed to provide the engineer with a high-powered computing capability, directly accessible, easy to understand and simple to operate. This computer is designed to be used in the department, in the lab, in the office, just as a slide rule is used: to obtain instantaneous answers to day-to-day problems — but at an increase in speed of several orders of magnitude.

This guide is written to give you, the reader, an understanding of the PDS 1020, and its built-in engineering interpreter, which permits extensive use of the machine without learning machine language. It may be used by anyone who needs a quick answer to a problem, not just by the professional programmer.

The interpreter is a computer program which allows simplified operation of the PDS 1020. Many design features have been incorporated into the hardware of the PDS 1020, to simplify the operation of the computer in the interpretive mode, and to make the interpreter more flexible and powerful.

Nevertheless, the interpreter should not be mistaken for the computer itself. The PDS 1020 is a full-scale digital computer, with a machine language of over 40 commands and extensive computing capabilities.

The interpreter, since it is a program, is flexible. Its capabilities and performance may be changed to suit individual applications.

The guide is divided into several sections. The first section deals with computers generally, explaining some of the fundamentals of what computers are, how they operate and what they can do.

The second section deals with the step-by-step method of operating the PDS 1020. When this section has been mastered, you are ready to operate the computer.

Section three discusses further capabilities of the machine, and the next section describes some fairly sophisticated uses. There is a separate section on how to use the paper tape reader and punch units. Finally, in the appendix, we included a summary of the information for quick reference.

Each of the sections is independent of the others, as far as operating the computer is concerned. It should be emphasized that this is a user's guide, designed to be a permanent reference, as well as learning text.

The real simplicity of operating the PDS 1020 Computer, and the extensive computing power at the engineer's disposal at the touch of a button, cannot be fully appreciated by reading this guide, or any other document. Only by sitting at the keyboard, entering a problem for solution, and getting an immediate result back, can these points be adequately realized.

1. INTRODUCTION

Fast Calculator or Thinking Machine?

In the dozen or so years since the first computer was commercially installed (and, incidentally, obsoleted) computers have entered almost every business and industry, revolutionized areas of technology and contributed scores of new words to the language. But while computers have multiplied like breeding rabbits, and every school child has heard of them, if not actually seen and touched one, the fact remains that few people know what they really are. In fact a computer myth has grown up, which holds computers to be omnipotent, diabolically clever and true thinking machines. A second view, held by the skeptics, considers the computer nothing more than a fast calculator with a fancy price tag.

Both views have a grain of truth in them, but both are inaccurate in that, like the blind men and the elephant, they consider only a single aspect of the whole. It is true that computers are capable of certain feats that humans can't perform. But there are many more areas where computers are helpless entirely, and certainly they can do nothing at all except for what they are told to do. On the other hand, comparing a computer to a fast calculator is like saying that a jet plane is nothing more than a fast ox cart — a device for transportation.

Essentially, the computer is a fast calculating machine - with two important differences: it has a memory, and is capable of making some logical decisions. These faculties enable the computer to do an enormous amount of numerical work, or solve any kind of problem that can be represented numerically, with great speed and efficiency.

The many diversified applications to which computers have been assigned

can be generally classed into three categories: data processing, process control, and scientific and engineering calculations.

In the area of data processing computers have taken over the function formerly performed by their elder cousins, the data processors or 'tab' machines, which utilized wired boards to instruct the machine in what it must do. The stored program computer merely substituted its flexible memory for the wires and boards, thereby increasing both its capacity and its ability to handle many diversified tasks.

The process control machine is a data processor of sorts, used to control and direct a set pattern of activities. These may range in complexity and scope from a relatively simple paint mixing operation to complete control of all activities within an oil refinery. In each case the computer is connected to a variety of instruments from which it receives information regarding the process in progress. This information is examined in accordance with pre-established instructions in the machine's memory, and decisions are made on the basis of the results of this evaluation. Finally, as a result of these decisions, signals are sent to control devices which make whatever changes are needed in the process in progress. All this is repeated at great speeds, so that the process is continuously monitored and controlled.

The engineering and scientific computer is the most recent arrival and the fastest growing among the three computer types mentioned. Its basic function is the solution of mathematical problems (or any problems for which mathematical equivalents or models can be found) in science, engineering, management, and almost every other field of human activity. Many of the technological achievements of the past few years are directly traceable to the computer and its ability to do in hours and days what may have taken months and years to do manually.

As computers grew larger, faster, and more expensive, the task of communicating with them became more complex and difficult. The reason is inherent in the basic concept of how computers work.

Computer Hardware

Basically, all computers of whatever size and price tag, have four functional parts: input/output devices, a memory, an arithmetic section, and a control section.

The input and output devices are the links between the machine and the outside world. Through them the computer receives instructions and data for its work, and through them it provides the answers which it produces.

The memory is used by the computer to store the instructions it receives, as well as the data on which it must operate. There are several types of computer memories, varying in speed, in size and in cost. All computer memories, however, have this in common: the ability to store information in some predetermined and organized manner so that it may be retrieved at will.

The arithmetic section consists of hardware which allows the computer to perform calculations. Like a calculator a computer has several registers which are used to hold the operators, the operands and the computed results. The most important of these registers is the accumulator, which usually holds one of the values used in a computation, and the result when the computation is done.

Finally, the control section, or, as it is sometimes called, the logic of the computer, coordinates the activities of the other components. Acting as a traffic cop, it directs information received from input devices to storage locations, retrieves it as needed, performs whatever operations are called for, and stores the results or communicates them to the user through the output devices.

Computer Software

All computers, the largest and the smallest, have these elements, and differ only in complexity, size, the number of input/output devices, the speed of computation, and, of course, in the price tag they bear.

To solve a given problem, the computer is given a series of instructions which are stored in its memory. These instructions, called a program, form the blueprint for solving a problem, much as an algebraic formula does: they can be used over and over, with different sets of numbers or data. To calculate X = 2a-b, for example, we must first give specific values to a and to b. Similarly, the computer could be given instructions to take the variable value, a, multiply it by two, subtract the value of the variable, b, and print on the typewriter the result, X. This program would be stored in memory; in a different section of memory we could store a hundred values of a and a hundred values of b, and add instructions to the program to tell the computer to evaluate each set in turn and print 100 X values on the typewriter. The many values of a and b are known as the data with which the program works, and are usually separate from the program itself. On the other hand, the number 2 by which a is multiplied remains constant, and is normally included in the program itself.

Programs are known generally as software, to distinguish between machine functions which are built into the machine hardware, and functions which the machine performs by using special purpose programs and which can be changed at will.

The main problem of communicating with a computer, is in stating the problem to be solved in some form which can be communicated to the machine. Since most computers use a language which is composed of special symbols, this is a task requiring some specialization and is frequently left in the hands of the programmers. Programmers normally have a good mathematical background, plus some specialized knowledge of the machines which they program. They are skilled and high price personnel. This is where the economics of computer utilization begin to play a part.

5 Milliseconds and 24 Hours Later ....

Consider, for example, the data processing computer, which handles as one of its routine tasks the preparation of the company payroll. Once the program for doing this has been written, it can be used over and over again, for years to come, with few, if any, changes. No matter how complex the program, or how much it costs to prepare, the effort is well worth while: the time expended can be saved many times over, and the cost can be amortized over a long period of time.

Or, consider the process control machine, in charge of some chemical process. Once set up, the machine can operate for years without a change in program. Only when the process changes does it require new software.

But the engineering and scientific machine has to solve a wide variety of problems, some of which are unique and are never repeated. An engineer

wants to examine the properties of a new wing design for a supersonic aircraft, for example. Once the answer is given, it is likely that the program for its solution will become totally useless. True, certain general programs may be written to evaluate some properties of all wings, but nevertheless many problems call for unique solutions, and unique programs to solve them.

Some computer manufacturers have attempted to solve the problem by offering extensive software libraries for use with their machine. This pre-packaged programming approach is not only costly but often futile. While solutions for a general problem area can be developed, the unique problem cannot be solved until the engineer is ready to place it in the computer. And when he is ready it is likely that none of the library programs will suit his exact needs.

In reality, many large companies employ a staff of programmers to do the translation job required to state engineering problems in computer language, despite the high cost of such a method. The fact is that many of the engineering and scientific tasks in today's technology can be solved only by computers, in the time span allowed for their solution.

In a typical installation, the engineer (or chemist,or statistician, or human factors researcher) goes to see the programmer, and explains to him the nature of his problem. The programmer proceeds to "code" the problem into the computer language. The coded program is then punched on cards or tape, media which can be rapidly communicated to the computer, and sent to the scheduler, who estimates the amount of time necessary to execute the program, and assigns an appropriate amount of machine time. Finally, the program is executed by the machine and the results are sent back to the engineer, or whoever originated the problem in the first place.

To anyone familiar with the complexities of the modern company, it is readily evident how long and tedious such a procedure can be, and how easily a minor mistake can slip by unnoticed, or creep in accidentally. When this happens, the procedure must start all over again. Even if the program already exists and does not have to be written especially, and top priorities are assigned to the program all along the way, twenty-four hours is the very least amount of time in which the engineer can expect an answer to his problem. Ironically, it may have taken the modern, second-generation, computer just a few milliseconds to do the actual computations, and less than a second to do the entire program, including all input and output.

Languages Berlitz Never Heard Of

It soon becomes evident that the problem of communications between departments is sometimes worse than the problem of talking to the machine. Many computer manufacturers, at the demand of computer users, set out to simplify the programming operations to the point where most of the users could do their own programming, leaving programmers only the more complex and sophisticated tasks. Out of these efforts came a variety of artificial computer languages such as FORTRAN (Formula Translator) ALGOL (Algebraic Oriented Language) COBOL (Common Business Oriented Language) PINT (Purdue Interpreter), and several others. These languages are problem oriented, meaning that they are intended for problems within certain fields, such as scientific calculation, and base their terminology on the terms and symbols common to that field. Thus, they are relatively easy to master for anyone who has the necessary background.

These languages, known as interpreters and compilers, serve the purpose of eliminating one step of the procedure enumerated above: instead of going to a programmer, explaining to him the problem to be solved, and having him write a program, the engineer writes his own program in FORTRAN and sends it to the computing center for execution. Nothing else changes, for the program must still be punched, scheduled, run and returned to the originator. Still, the saving in time is considerable, and more important, the engineer, or the specialist within the field, provides his own methods of solving the problem, without having to rely on a programmer who may be proficient in mathematics and ignorant in the particular field of specialization to which the problem relates.

Instant Answers

Compilers and interpreters have extended the range of problems to which it is practical to apply a computer, but for many problems the computer is still out of reach. It is impractical to write a program even in FORTRAN, and wait a day or two for an answer, if a slide rule and calculator will provide the solution in two or three hours. Ideally, if the engineer-user had direct access to a computer, these problems could be solved in minutes or seconds.

Up to now it has generally not been economical to supply engineers with a computer to which access is directly available. The main reasons for this are both the high price tag of computers and the difficulty of communicating with them directly. It is usually cheaper to prepare the program on punched cards or tape, employing special personnel and equipment to do so, than it is to communicate directly with the computer through a typewriter or keyboard, taking up its valuable calculating time. To go through such a procedure for a relatively simple problem is like flying a jet plane across town - the speed of the jet is more than offset by the long trip to and from the airport, take-off and landing time, etc. Thus, a gap exists between those problems which can be readily solved by manual means, using a calculator, a slide rule or merely pencil and paper, and those problems for which computer solutions become efficient and economically feasible.

The PDS 1020 Computer is designed to bridge this gap, and to provide a problem solving tool, which may be used directly in the engineering department, in the lab, at the drawing board, or even on location. Simple to understand and to operate, the 1020 computer is simpler than a slide rule to learn, yet is a real computer with extensive powers, capable of problem solving at electronic speeds.

The 1020 computer is equipped with a built-in, custom-made interpreter. Custom-made in the sense that the functions performed by the interpreter can be adapted to the individual needs of the user. Built-in in the sense that it operates through a push-button keyboard, clearly labeled with the function of each key. The combination of these two features permits the user to perform highly complicated calculations at the push of a button.

A paper tape punch is provided for permanent storage of programs and data. A paper tape-reader is also included and may be used as an input device as well as a fast keyboard, where desired, feeding instructions to the computer.

A typewriter is used to furnish the results of computations.

Finally, the 1020 computer bears a price tag which makes it practical and economical to use within the confines of the department, so that the user

can directly approach it, enter his problem and receive an instant answer.

This guide describes the capabilities of the 1020 computer and how to use its built-in interpreter. Since the computer is easy to operate, the guide should be easy to read; formal language, technical vocabulary and elegant phrasing have been sacrificed for the sake of simplicity and clarity.

2. GENERAL DESCRIPTION

What Can it Do

The 1020 Engineering Interpreter is a program written for the 1020 computer and its function is to allow the engineer or scientist to operate the 1020 computer without having to learn machine language.

There are three ways to operate the 1020 computer, using the Engineering Interpreter:

1. Operating the computer one step at a time. The user enters data through the keyboard or tape reader and performs desired calculation on this data. The interpretive keyboard of the 1020 allows a variety of multistep algebraic functions to be performed at the touch of a button. Square root, exponentiation, sine, cosine and arctangent values, and the natural logarithm of a number may be arrived at in this way. It should be emphasized that these functions are part of the interpreter program, and may be replaced by others, if other functions are desired by the user. Thus, any mathematical evaluation which is frequently performed in a given application, may be included in the interpretive keyboard functions, and executed by pushing a button. The procedure for solving complicated problems by this method is fully described in the following pages, under the heading "Step-by-Step Operation."

2. Operating the computer by program. Essentially this procedure is no different than the step-by-step operation, but the computer is instructed to retain, or remember, the steps for solving a given problem. Thus the problem is solved again and again, automatically, using new data for each solution. The procedure for retaining the problem solving steps is shown under the heading "Retained Operations."

3. The paper tape reader can be used to enter instructions or data, and to operate the computer as a fast keyboard. This operation can be independent or in conjunction with a retained program or keyboard instructions.

You can operate the computer, using any of these methods to solve a great variety of problems, quickly and efficiently, without having to learn machine language. In fact, the operation of the computer can be learned in a matter of an hour or two.

The Equipment

The 1020 computer is compact in structure, occupying no more space than an office desk. No installation is required beyond plugging it into an ordinary household outlet.

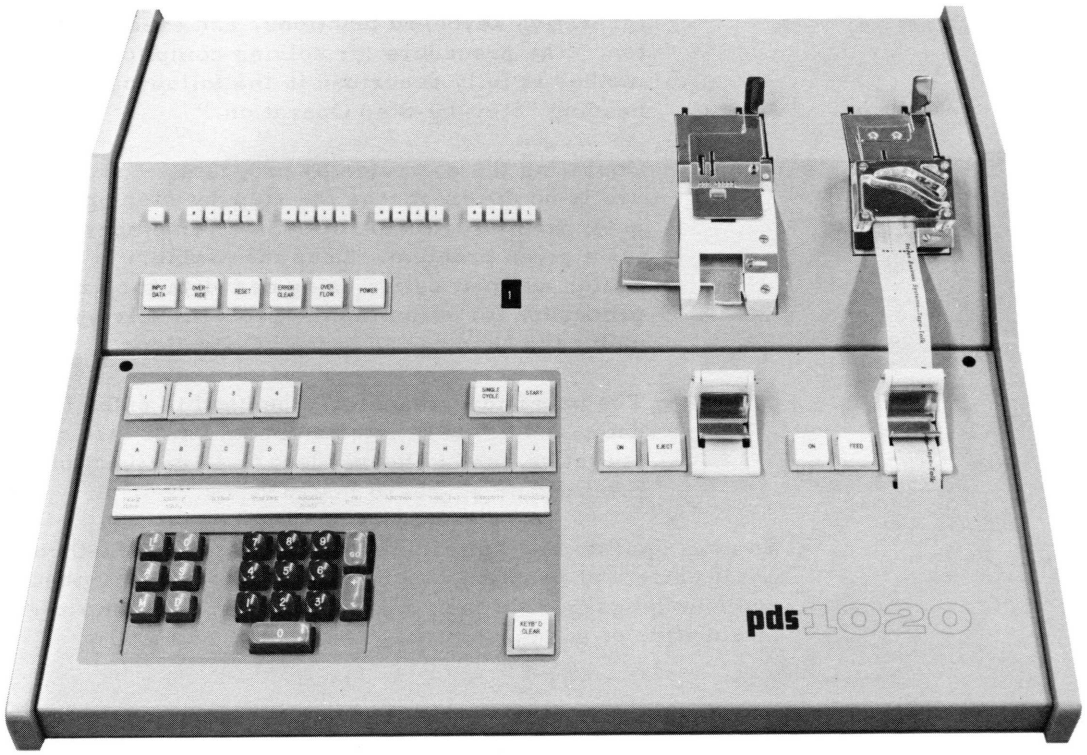
The operator, sitting at the computer, has within easy reach all three elements used to communicate with the machine: to his left is a typewriter used by the computer to print out answers; to his right is a tape reader and tape punch, which are used for fast input and output, and for making a permanent record of a problem solving sequence; to the left of the tape equipment is the keyboard, which is used by the operator to enter data and perform calculations.

The keyboard itself contains a variety of controls, indicators and pushbuttons. Some of these are essential to the operation of the computer in

the interpretive mode, some are used only in the machine language mode, and some should only be used as trouble shooting devices. The functions of these controls and indicators will be described along with the operating procedure; those controls not described are not essential to the operation.

In summary, the 1020 computer is suited to applications which are time consuming to handle manually yet not complicated enough to justify the use of a large computer. By using the 1020 Engineering Interpreter, the user can learn in a short time to solve his own problems on the computer, without resorting to programmers, keypunch operators, or other middlemen. The computer is desk size, and plugs into an ordinary outlet, permitting its use within the engineering department, or any other location which is convenient to the user.

The actual operation of the computer is described on the following pages.



3. STEP - BY - STEP PROCEDURE

The operation of the computer may be broken down into three basic stages:

1. INPUT - This consists of entering all the data to be used in a particular calculation, and storing it in memory.

2. COMPUTE - The computer uses instructions to operate on the data and compute results.

3. OUTPUT - The results are typed on the typewriter or punched on tape.

STAGE 1. INPUT

All numbers are entered in floating point format, meaning as a multiple of a power of ten. This form of notation, common in many engineering applications, was chosen to simplify the manipulation of large numbers.

A floating point number can be written in many different ways, since the decimal point may be placed at will, and the exponent adjusted accordingly. As an example, the number

$$123.45$$

may be written in floating point notation as any of the following combinations:

$$1.2345 \times 10^{2}$$

$$.12345 \times 10^{3}$$

$$12345 \times 10^{-2}$$

The value of the number remains unchanged, since the exponent is adjusted to suit the decimal point (hence the term floating point).

All numbers used as input to the 1020 computer should be adjusted to reflect a fraction and a power of ten only. Thus, in the above example, only the format

$$.12345 \times 10^{3}$$

is acceptable for input. The number 10 is assumed and does not have to be entered, and only its power is necessary. Thus, the number would be entered as

$$.12345 + 3+ .$$

The general format of numbers used as input to the 1020, is

$$.ddddd+ \; n+$$

Where ddddd are any decimal digits, and n is a power of 10.

The sign of the fraction and the sign of the exponent, both must be entered following their magnitudes.

Up to 8 digits may be entered as the fractional part of the number, not counting the decimal point or the sign. The exponent must be in the range -99 to +99.

If more than 8 digits are entered, only the last 8 will be used. This affords a quick method of correcting errors in entering numbers. For example, we want to enter the number

.123 + 3+

but instead we enter

.124

This may be rectified by continuing to enter sufficient zeros to discount this number, and following these with the corrected entry:

00000123+3+

The entire number entered is, therefore

.12400000123+3+

Only the last 8 digits are effective, so that the number remaining in the accumulator is

.123+3+

since the computer ignores leading zeros. Note that this is always the case and that the number .00000123 must be entered as .123 + 5-, to obtain correct results.

Numbers entered from the keyboard are automatically placed in the accumulator, and the computer awaits further instructions. A storage instruction may now be executed, placing the number in one of the 100 locations in memory reserved for that purpose.

To store a number, the following sequence of keys is used:

1. Depress the key marked CPY.
2. Depress the numbers keys for the desired location. For example, key 2 and key 3 for location 23.
3. Depress the key marked GO.

The number will be stored in location 23, where it will remain until a new number is placed in that location. Note, however, that until the next number is entered from the keyboard, the number still resides in the accumulator as well. It was merely copied by the storage operation, not destroyed.

When the number has been stored, a second number may be entered through the keyboard, following the same procedure. All numbers necessary to perform a given operation must be entered in the same way, and stored in selected memory locations.

STAGE 2. COMPUTE

Arithmetic operations call for at least two values: a number is added to another number, subtracted from another number, multiplied or divided by another number. In the computer one of these values is in the accumulator and the other is in a data storage location. There are 100 such locations numbered 0 - 99, and they are referred to as scratchpad memory or scratchpad locations, since they serve the same purpose as the engineer's scratchpad, where he may note the values he is working on,

write down the intermediate results, etc.

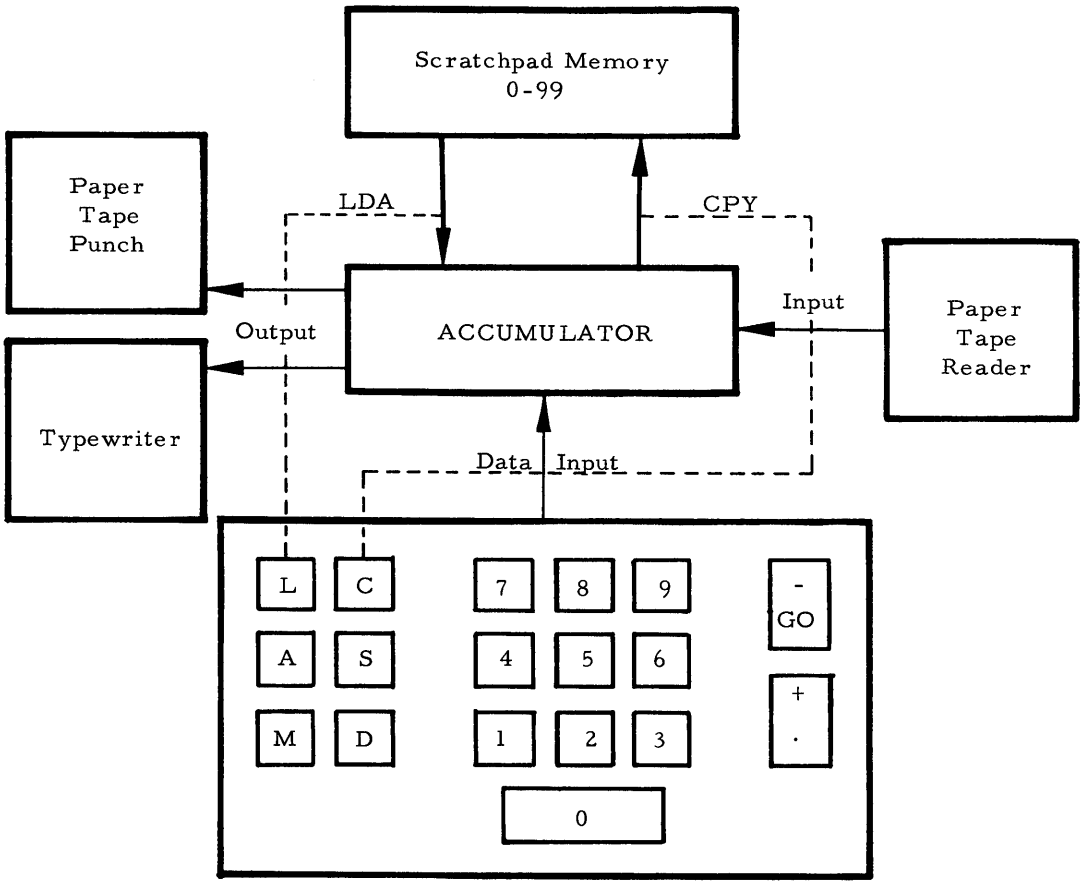
The value in the accumulator got there in one of three ways:

1. It was input from the keyboard (or tape)
2. It was recalled from scratchpad storage
3. It was already in the accumulator as a result of a previous computation.

Note that when a computation is completed, the result is always in the accumulator, and may be used as a value in the next operation. For example, if we calculated A + B in one operation, we may now divide by C to arrive at $\frac{A + B}{C}$.

Figure 1 shows the flow of data into and out of the accumulator.

FIGURE 1.



KEYBOARD

Data Flow In The PDS 1020 Computer

Note that input and LDA change the contents of the accumulator; Output and CPY do not change the contents of the accumulator.

The four basic arithmetic operations are performed by depressing the key marked with the function to be performed, the memory location where one of the two data values operated on is stored, and the GO key.

To simplify notation, the procedure will from now on be written as

OPR XX GO

Where OPR represents the key of the particular function (ADD, SUB, MPY, DIV), XX represents the memory location number, and GO represents the GO key.

The accumulator must always contain one of the numbers being used in any arithmetic operation. If the number is not already there, as a result of keyboard entry or a previous calculation, the accumulator has to be loaded with the desired number, from its storage location in memory. The procedure is

LDA XX GO

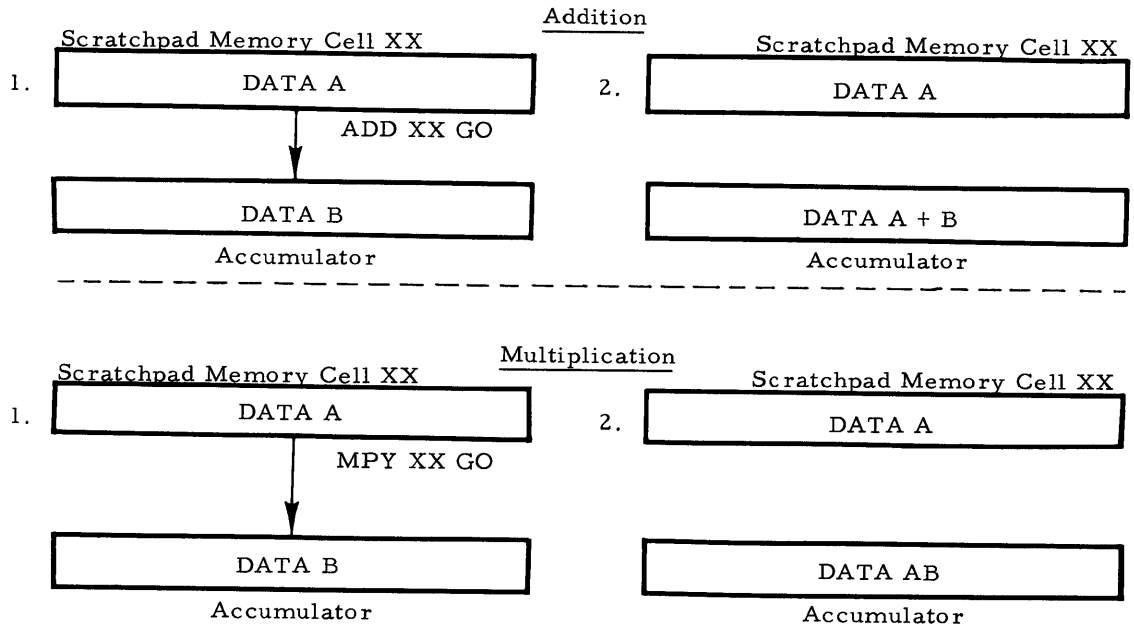
where XX is the memory location containing the number to be loaded.

To add or multiply, the same procedure is used:

ADD XX GO
MPY XX GO

The number in XX will be added to or multiplied by the number in the accumulator. The result will remain in the accumulator. The values in memory locations XX, remain unchanged.

FIGURE 2.



The same procedure is also followed for subtraction and division, but it should be remembered that the sequence

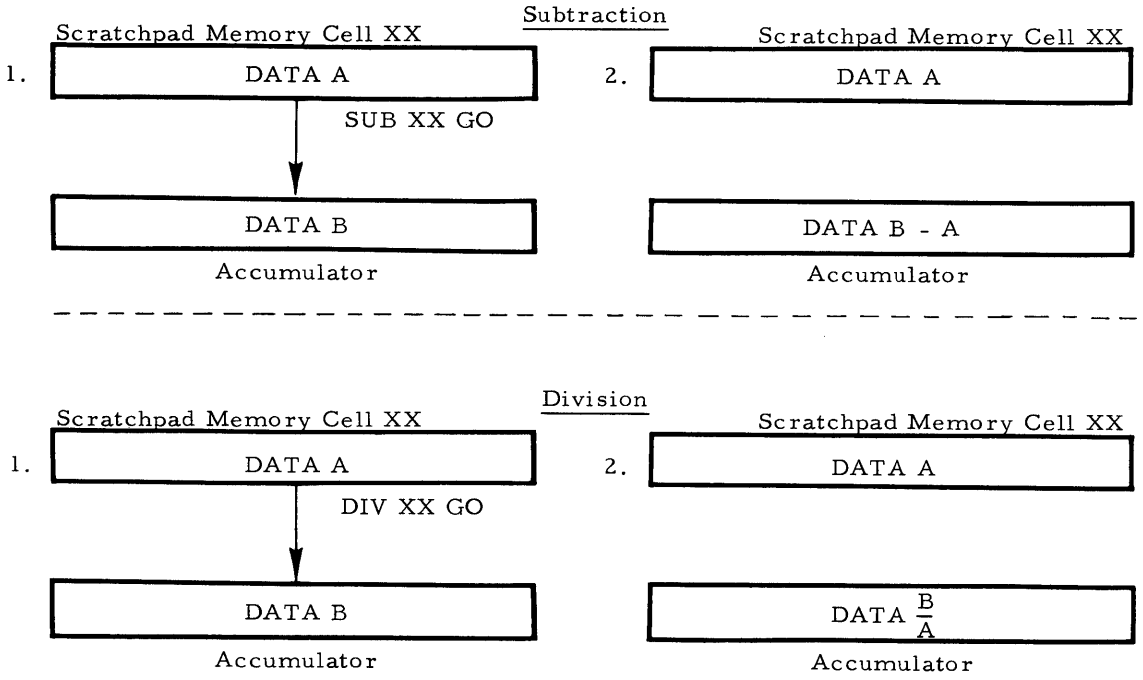
SUB XX GO

means subtract the number in XX from the number in the accumulator. Similarly the sequence

DIV XX GO

means divide the number in the accumulator by the value contained in XX. The value in XX remains unchanged, and the result of the computation is in the accumulator.

FIGURE 3.



In addition to the four basic arithmetic operations it is possible to perform any of the special functions included in the engineering interpreter. The standard functions in the 1020 engineering interpreter are sine, cosine, exponential, logarithm, square root, and arctangent. Any or all of these functions may be replaced with others, more suitable to a particular application, if the user so desires. The standard functions are discussed here; special functions, if used, would conform to the same general procedures.

All functions use the value in the accumulator only - a second value is not necessary. To perform a particular function, the value to be operated on is loaded into the accumulator, and the function key corresponding to the desired operation is depressed. The results of all functions replace the previous contents of the accumulator.

SINE: The value in the accumulator is assumed to be in radians, and must be less than 30,000 radians. This value is considered the argument and its sine is calculated. The result, in radians, re-

places the original value of the accumulator.

COSINE: The cosine of the value in the accumulator replaces the contents of the accumulator.

EXPONENTIAL: e is raised to the power contained in the accumulator. This power should be less than 9.2, so that the result will be less than 10,000.

LOGARITHM: The natural logarithm (base e) of the absolute value in the accumulator is calculated and replaces the contents of the accumulator. If the accumulator contains zero before the LOG key is pushed, the computer will halt. (See Trouble Shooting Procedure.) The accumulator should contain a value less than 10,000.

SQUARE ROOT: The square root of the absolute value of the number in the accumulator is calculated. The result is placed in the accumulator, replacing the previous contents.

ARCTANGENT: The arctangent of the value in the accumulator is calculated; if the value equals x then -

$$\text{if } x \geq 100, \text{ ARCTAN } x = \left[\pi/2 - 1/x\right] \text{ with the proper sign.}$$

The result is in the accumulator, when the function is done.

Note that these functions do not require the use of the GO key. The operation is executed as soon as the function key is depressed.

STAGE 3. OUTPUT.

So far, numbers have been entered and stored in memory, and arithmetic and mathematical operations have been performed, using them. But the results, final or intermediate, are still in the computer, and must be typed out on the typewriter to become meaningful to the user.

Output is from the accumulator. As in the case of numbers entered into the machine, which go to the accumulator from the keyboard and are then transferred to specified memory locations, so in output the values must be loaded into the accumulator and then typed out. In most cases, however, this does not require any special operations, since the results of arithmetic operations and special functions are automatically placed in the accumulator. All that is required to type out these answers is the instruction

CPY +

Note that depressing the + key rather than the GO key after pushing CPY, gives the CPY key an entirely new meaning: Copy a value from the accumulator on the typewriter.

The value is typed out as an eight digit fraction followed by a sign, and a two digit exponent, followed by a sign. The general format is

.xxxxxxxx + (or -) YY + (or -)

where X is any decimal digit, and YY represents a power of ten. The number

123.45

would be typed as

.12345000+ 03+

A carriage return may be initiated from the keyboard by the sequence

DIV +

This is convenient during step-by-step operations, eliminating the need to work the typewriter keyboard, but is absolutely essential in programmed operations.

The instruction

SUB +

will punch the contents of the accumulator on paper tape. The uses of this instruction and of paper tape equipment will be shown later in this guide.

The step-by-step procedure is sufficient for the solution of any one time calculation for which an answer is desired in a hurry.

The following example shows the use of the step-by-step technique and its flexibility.

Where the problem must be solved more than once, however, or for computations involving iteration of several steps, the retained approach should be used. Nevertheless, the computer can be used economically for solution of problems at this stage. The main point to remember is that the solution is arrived at by calculating it step-by-step in much the same way as if pencil and paper or a slide rule were used. When you have mastered this concept and understand the following example, you are ready to use the 1020 computer.

EXAMPLE 1

Problem: Calculate X where

$$X = \frac{A+B}{C+D} \sqrt{\frac{1}{F}}$$

Method: Assume that the five values listed below are given and that nothing else has been stored in memory, so that the locations 1 - 5 may be used for storing these values.

To avoid confusion it is desirable to list the scratchpad locations which will be used in the solution prior to actually solving the problem. In this way the user knows where his data is located, and can easily use it when needed.

Assume then the following values and storage assignments:

A = 173.86 and will be stored in location number 1.
B = 13.281 and will be stored in location number 2.
C = 15.95 and will be stored in location number 3.
D = 10.953 and will be stored in location number 4.
F = 25.971 and will be stored in location number 5.

In addition, locations numbers 6 and 7 will be used for temporary storage of intermediate results.

The solution is arrived at through the three stages as described above.

STAGE 1.	.17386+ 3+	
	CPY 1 GO	A is entered and stored in location 1.
	.13281+ 2+	
	CPY 2 GO	B is entered and stored in location 2.
INPUT	.1595+ 2+	
	CPY 3 GO	C is entered and stored in location 3.
	.10953+ 2+	
	CPY 4 GO	D is entered and stored in location 4.
	.25971+ 2+	
	CPY 5 GO	F is entered and placed in location 5.
		All values are now in their assigned places.
	.1+ 1+	A 1 is placed in the accumulator.
STAGE 2.	DIV 5 GO	The value 1/F is now in the accumulator.
	SQUARE ROOT	The root is extracted and placed in the accumulator.
	CPY 6 GO	The root is temporarily stored.
	LDA 4 GO	Load accumulator with D.
COMPUTE	ADD 3 GO	Add C.
	CPY 7 GO	Store (C+D) temporarily.
	LDA 1 GO	Load accumulator with A.
	ADD 2 GO	Add B. (A+B) is now in the accumulator.
	DIV 7 GO	Divide (A+B).
	MPY 6 GO	Multiply by the root of 1/F.
STAGE 3. OUTPUT	CPY +	Type the answer on the typewriter.

4. RETAINED OPERATIONS

The step-by-step procedure is a convenient and fast way to get answers to some problems which cannot be readily solved manually or with the aid of a calculator. Still, there are problems which require additional capabilities, and for which solution by the step-by-step method becomes tedious and uneconomical. Such problems may be solved through retained operations. Furthermore, where the problem is not unique, even the problems described above can benefit from the capability of the computer to retain the steps involved in the solution so that they may be solved over and over, using new sets of data each time.

A program is merely a sequence of steps involved in solving a problem. It corresponds to the second and third problem solving stages described above, namely the computing stage and the output stage, and makes certain provisions for the first stage to input data.

In a sense a program is like an algebraic formula: It establishes a general method for solving a given problem, without being concerned with the actual data values involved. The program, like the formula, does not produce a solution until the general values are replaced with specific numeric values. Thus, once a program is stored in memory, it serves to solve the problem as many times and with as many sets of data as the user desires.

If the program is one which is used often, it may be punched on paper tape, and then entered through the paper tape reader, eliminating the need for entering it manually through the keyboard.

The program is entered through the keyboard, just as in the step-by-step procedure. There are some new commands and functions which may be used in programmed operation.

RETAIN:

This command sets up the programmed operation: It instructs the computer to store all succeeding commands in its memory, until it receives a second RETAIN command.

Six different programs, or sets of instructions, may be retained by the computer at any one time. These may be entirely independent or may interconnect, as will be seen later. To identify programs they are numbered from one to six, and the number is entered along with the command to retain. The general format of the RETAIN command is therefore:

RETAIN N +

The RETAIN button is pushed, then a number key, from one to six, then the + key. To terminate the Retain mode, the RETAIN key is depressed a second time.

In this way the instructions

RETAIN N + ... RETAIN

serve as brackets, enclosing the instructions to be stored in memory.

EXECUTE:

The execute command causes the computer to perform the operations in the retained program. Like the RETAIN command, EXECUTE must be

followed by the number of the program to be executed since there may be up to six programs stored in memory. The general form is

EXECUTE N +

where N is any number from one to six.

It is permissible to include an EXECUTE command within the retained program, provided that it refers to a different program. For example, in program number 1 an EXECUTE 2 may be given. When program 1 is executed it will go on to execute program number 2 at the point where the command was given. This capability is very important, and adds a great deal of flexibility to the computer. Examples of the usefulness of this feature will be shown later.

Since the retained program may contain many steps, the user may wish to see it typed out and make sure that no errors in entry occurred. This can be done by means of the command

EXECUTE N GO

Which will then type out program N on the typewriter. The typeout will be in abbreviated format, and will include each step number and the command. Commands are abbreviated so that ADD becomes A, SUB becomes S, MPY becomes M and DIV becomes D. In addition, the special keyboard functions are shown in special codes (see table in appendix). The memory location involved is also typed out so that the typed instruction may look like

001 A025 -

meaning at step number 1 add the contents of memory cell 25 to the contents of the accumulator.

INPUT VARIABLE

The retained program contains only instructions for solving a problem. Data must be entered separately. This is, of course, the main advantage of the retained program, since it allows the same instructions to be used with many different sets of data. A special instruction is used within the retained program to prepare the computer for receiving data inputs. This instruction is entered through the interpretive keyboard by depressing the INPUT VAR key.

During execution of the retained program, when the computer reaches the input instruction, it will stop and allow the user to input a data value. The value is loaded into the accumulator, as in the step-by-step procedure, and may then be stored by means of a CPY instruction. The CPY instruction, however, should be part of the retained program, since the computer will not accept any instructions from the keyboard while executing a retained program.

To input the values of the variables in Example 1, the following sequence of instructions could be executed in the retain mode:

```
INPUT VAR
CPY 1 GO
INPUT VAR
CPY 2 GO
INPUT VAR
CPY 3 GO
INPUT VAR
CPY 4 GO
INPUT VAR
CPY 5 GO
```

This sequence in the retained program takes the place of Stage 1 of the step-by-step procedure. Note that even though provisions have been made for entering data, no actual data has as yet been entered. When the retained program is executed, the computer will reach the first INPUT VAR instruction and stop. The data will be entered at that time. No storage instructions will be required, since these already appear in the program. The user can enter five data values in succession, and these will be stored in the specified locations. As soon as the last value is entered the computer will go on to calculate and print out the solution, following the steps in stages two and three of the example. When the program is done and the solution printed out, the user need only to depress the EXECUTE button, along with the number of the program and the + key and he is ready to enter five new values, and calculate the problem over again.

Operating the computer in the Retain mode is thus similar to using the step-by-step procedure with this difference: instead of executing each instruction as it is received from the keyboard, the computer retains all instructions in memory, and executes them only after receiving an EXECUTE command.

Six different programs may be retained in memory at any one time, containing a total of over 450 instructions. This figure does not include the 100 scratchpad locations used for data storage and is based on a 1020 computer with a 2048 word memory. If additional memory is installed, the number of instructions is increased by the number of added memory locations.

The instructions may be arbitrarily divided among the six retained programs to suit the user. Thus, one program could contain 400 instructions and the other five the balance, or just a single program may be used, utilizing all available instructions.

At this point you have sufficient information to use the computer either in the step-by-step procedure, or in the Retain mode, for all applications requiring linear solutions: i.e., computations which proceed from beginning to end in a sequential order and which do not require sequences of instructions to be repeated.

More sophisticated use of the computer is possible, using its capabilities for making logical decisions to repeat segments of the program. These capabilities are discussed in the following chapter.

5. TESTING, JUMPING AND LOOPING

The computer normally executes instructions in the order in which they are received. In the Retain mode the instructions are stored sequentially and when the time to execute them comes they are performed in the same order in which they were entered. Frequently, however, it is advantageous to disrupt this orderly procedure and to execute a different sequence of instructions. This may be accomplished by means of a transfer to the desired sequence.

Transfers are of two kinds: an unconditional transfer is an instruction to perform a different sequence of instructions, located at some other location in memory. This transfer is not concerned with the instruction just performed or its results. A conditional transfer, on the other hand, provides for a test of a value in the accumulator: if the value corresponds to a given condition (i.e., it is negative or positive, or smaller, greater or equal to some other value), then the transfer occurs. Otherwise the computer continues in its sequential executions. Both types of transfer are extremely useful and provide many shortcuts in problem solving. Some uses are shown below.

When a transfer is used in a program, it is important to number each of the problem solving steps sequentially, so that the transfer may refer to the new instructions to be executed by step number.

For example, suppose that in a given program ten instructions are to be executed in order and then a jump is required to do the last five instructions over again. The steps should be numbered from one to ten and a jump instruction to step five should then be given.

In the 1020 engineering interpreter, the unconditional jump instruction consists of depressing the following sequence of keys:

7 XXX GO

where XXX is the step number to which the transfer is required. In the example above the jump instruction

7 5 GO

would effect a transfer to the fifth step. Note that the step number may be entered without leading zeros.

The conditional transfer is effected by depressing the TEST JUMP key on the interpretive keyboard. The general sequence is

TEST JUMP N +

where N is the step number to which transfer will occur. The condition set by this instruction is that if the number in the accumulator is negative, the transfer will take place; if not, the transfer will not be effective and the computer will proceed with its sequential execution.

This instruction can easily be used to set up other conditions as well. Suppose that at step 25 in our program it is desired to jump to step 40 if the number in the accumulator is positive or zero and to continue sequentially if the number is negative. The sequence

STEP 25 TEST JUMP 27+
STEP 26 740 +
STEP 27 Continuation

will accomplish this purpose. If the accumulator holds a positive number, the test will fail and the computer will execute step 26, jumping to step 40. If the accumulator contains a negative number, the jump to location 27 will continue the sequential execution.

EXAMPLE 5, on page 29, illustrates the use of jump and test jump instructions to calculate y = AX + B where X is automatically incremented several times until an upper limit is reached.

There are two common ways to use transfer instructions: one is in writing sub-programs, and the other is in constructing program loops. A third way, also commonly used, is often overlooked since it is an implied transfer. The EXECUTE command is in essence a transfer to the beginning of a retained program.

SUB-PROGRAMS

Frequently it is necessary to use a given sequence of instructions many times throughout a program. For example, assume that the program requires a certain value X to be evaluated according to the formula

$$X = 2\sqrt{R} \ .$$

It is easy enough to write a sequence of instructions to solve this formula, but if the formula is frequently used by the program, using different values of R, then it is more economical to write a sub-program to solve the formula, and transfer to it whenever X must be calculated. At the end of the sub-program another transfer instruction is given, returning to the main program.

The main program can be set up to place the current R value in the accumulator prior to the transfer. The sub-program then calculates X, leaving the X value in the accumulator and returning to the main program. The main program now goes through whatever calculations are necessary, using X, then places another R value in the accumulator, transfers to the sub-program, and gets a new X value to work with.

The sub-program can be relatively simple, as shown above, or very complicated. The main criterion for writing a sub-program is the number of times the particular sequence is used by the program.

A simple way to write a sub-program is to use the capability, mentioned previously, of cross referencing retained programs. Assume that the main program is number one, and the sub-program is number two. The following sequence illustrates how these programs interact. All instructions except retain and execute are represented by dots, to simplify the example.

The sub-program is executed three times in this example. Each time it is executed one instruction is issued (EXECUTE 2 +). The program itself, as shown by the dots, occupies five instructions. To write it three times would require 15 instructions. Thus, a considerable saving in memory space is effected as well as a saving in the user's time required to enter the separate instructions. Normally, sub-programs would occupy many more than five instructions, and the saving can be even more significant.

EXAMPLE 2

```
RETAIN 1 +        Beginning of Main Program
    .
    .
    .
    .
    .
    .
EXECUTE 2 +       The entire sub-program is executed
    .
    .
    .
    .
    .
EXECUTE 2 +       The entire sub-program is executed
    .
    .
    .
    .
EXECUTE 2 +       The entire sub-program is executed
    .
    .
    .
    .
RETAIN            End of Main Program
RETAIN 2 +        Beginning of sub-program
    .
    .
    .
    .
    .
RETAIN            End of sub-program
EXECUTE 1 +
```

Note that the instruction EXECUTE 2 + is issued before program number 2 is entered in memory. This is permissible in this instance, since the instruction EXECUTE 1 + is given after the sub-program has been entered and stored. Consequently, by the time the computer is ready to execute the instruction EXECUTE 2 +, program number 2 is already in memory.

A sub-program which is separately stored under a new RETAIN command is entirely independent of the original program. It may be used by more than one program if desired. Assume, for example, that a particular engineering application needs a tangent of a number calculated for use in various problems and formulas. The program for calculating the tangent could be written as a separate program, (say program number 6), and be used by five other programs in the machine by using the EXECUTE 6 + command.

When the sub-program is through, it will automatically return to the main program and resume further execution there. In the illustration, the number 2 program is executed and the computer then returns to the number 1 program, executes it until it comes to the next EXECUTE 2 + instruction, etc.

A second type of sub-program may be written simply by issuing a jump instruction, either conditional or unconditional, which transfers to the first location of the sub-program. This type of sub-program is essentially part of the main program, but may be executed many times at dif-

ferent points in the program. The return to the main program from a sub-program of this nature is not automatic, and must be provided for by writing special transfer instructions.

LOOPS

A loop is a sequence of instructions executed repeatedly a given number of times or until a certain condition is met. To terminate a loop by means of a condition is a simple matter of setting up the condition and issuing a conditional jump instruction. To terminate the loop after a given number of times, a counter must be set up internally to monitor the number of times a loop has been executed.

Looping is one of the most useful of computer functions since it allows iterations such as calculating a function for various automatically incremented values, repeating a calculation a specified number of times, etc.

Example 3 shows a program segment where you want to repeat steps 50 - 55 a given number of times, say N.

Set up a counter in any convenient scratchpad locations, say 0 and 1. Into cell 0 copy N-1 and into cell 1 copy a 1. Beginning with step 56 load the counter into the accumulator, subtract 1 from it, copy it back for future use and test it by means of a TEST JUMP instruction. If the number is positive, instruction 60 is executed, transferring to step 50 to do the loop again. When the counter is decremented after the Nth execution it will have the value -1, and the test jump will transfer to the continuation at step 61.

EXAMPLE 3

Step Number	Instruction	
	RETAIN 1 +	
1..	.	
.	.	
.	.	
50	.	Beginning of loop
51	.	
52	.	
53	.	
54	.	
55	.	End of loop instructions
56	LDA 0 GO	The counter is loaded into the accumulator
57	SUB 1 GO	1 is subtracted from the counter
58	CPY 0 GO	The decremented counter is stored
59	TEST JUMP	A jump to the continuation
60	750	An unconditional jump to the loop starting location
61	PROGRAM CONTINUATION	

EXAMPLE 4 shows the other instance of a loop application, where the loop is executed repeatedly until a condition is met. Suppose you want to compute y, where $y = ax + b$, and x is incremented by a constant until an upper limit is reached. The upper limit of x therefore constitutes the condition: if this limit has not been reached, the loop is executed again, using a new x value. If the limit has been reached, the program is through.

Assume that a and b are constant, and that their values are 5 and 67. 139 respectively. y is computed for x values starting at . 1 and up to 1. 0,

in increments of .1. The values are entered into the computer using the step-by-step technique, and are stored in memory in storage location 1 - 5 as shown in the table below:

	Loc. No.	Symbol	Value
$x_1 = x$ initial value	1	a	.5+ 1+
$\delta x = x$ increment	2	b	.67139+2+
$x_F = x$ final value	3	x_1	.1+ 0+
	4	δx	.1+ 0+
	5	x_F	.1+ 1+

When these values have been entered, the Retained program computes y for the various x values and types out x and a corresponding y value.

EXAMPLE 4

Step	Operation	Comment
	.5 + 1+	a is entered into Accumulator
	CPY 1 GO	a is stored in location 1
	.67139 + 2+	b is entered
	CPY 2 GO	b is stored
	.1 + 0+	x_1 is entered
	CPY 3 GO	x_1 is stored
	CPY 4 GO	The same value is used as δx and stored in location 4
	.1+ 1+	x_F is entered
	CPY 5 GO	x_F is stored in location 5
	RETAIN 1+	The first RETAIN is not a numbered step
01	LDA 3 GO	Accumulator is loaded with x
02	DIV +	Typewriter carriage return
03	CPY +	x is typed out
04	MPY 1 GO	x is multiplied by a = ax
05	ADD 2 GO	b is added, yielding y(y = ax + b)
06	CPY +	y is printed out
07	LDA 3 GO	x is reloaded into the accumulator
08	SUB 5 GO	$x-x_f$ If $x < x_f$ the result is negative
09	TEST JUMP 11+	If $x < x_F$ step 11 is executed next
10	715 GO	If $x \geq x_F$ the Test Jump will not be executed, and this instruction will transfer unconditionally to 15, and terminate the program
11	LDA 3 GO	x is reloaded into Accumulator
12	ADD 4 GO	δx is added to x
13	CPY 3 GO	The result is stored as the new x value
14	72 GO	Jump to step 2 and repeat
15	RETAIN	When $x \geq x_F$, step 10 is executed jumping to this instruction and terminating the program
	EXECUTE 1+	This instruction will execute the program

It is frequently desirable to execute loops using new sets of data for each execution. This may be done by including INPUT VAR instructions at the beginning of the loop. Data can then be entered through the keyboard for each successive iteration. Sometimes it is desirable however, to enter all data at the beginning and let the machine seek it in successive storage locations. To accomplish this it is necessary to change the storage location from which the accumulator is loaded at the beginning of the loop.

A special instruction is used for this purpose.

The instruction

8 XXX GO

is used to modify an address of an operand.

ADDRESS MODIFICATION

Memory storage locations, or cells, are numbered consecutively for identification purposes, as explained earlier. The number of a memory location is referred to as its address, since it serves to locate that particular cell. An address carried by an instruction, such as ADD 25 GO, is called an operand address, since it refers to a number which will be used by the computer: added, subtracted, copied into memory, loaded into the accumulator, etc.

Suppose you have 50 numbers stored in consecutive scratchpad locations, starting at location 50, and that you want the sum of all these numbers. You can do this by entering 50 instructions: ADD 50 GO, ADD 51 GO ... ADD 99 GO. This would do the job but it is obviously time consuming and tedious.

By modifying the address of the instruction ADD 50 GO, you can do the entire operation in 5 instructions:

EXAMPLE 5

```
      RETAIN 1 +

1     ADD 50 GO
2     84 GO
3     71 GO
4     0101 +
5     RETAIN
```

Instruction number 2 means "modify an operand address according to step 4." Step 4 means "add 1 to the operand address in step 1."

Instruction 1 will initially add the number in cell 50 to the accumulator. Instruction 2 will now change the operand address to 51, so that when instruction 3 is executed, transferring back to 1, the instruction reads ADD 51 GO. This process is repeated until ADD 99 GO is reached. Since this is the last scratchpad location the computer will stop.

The savings in time and effort, using this procedure, are evident.

The general form of the instruction to modify is

```
      8 XXX GO
      .
      .
      .
      .
      .
      .
XXX . SS II +
```

where XXX is the step number containing the information SS II, meaning add II to the operand address in step SS. Note that step XXX should be in an area of program which is never reached, as for example, following an unconditional jump. SS II is coded information, not an instruction. If it

is reached by the computer in its sequential execution, the computer will halt. The modification instruction is very useful where you work with tables stored in the computer (see example in appendix).

Note that when the program in Example 5 is done the instruction at step 1 reads ADD 99 GO. If the program is to be executed again, this should be changed again to read ADD 50 GO. A special instruction is provided for this function, which is called initializing the address.

Initializing the address is simply setting it to a specified value during the operation of the program. In the example it is necessary to set the address of the ADD instruction to 50, before new data is entered, so that it may be accurately modified by the address modifier.

Initializing and modifying the address during the program allows you to execute the program with many sets of data, so long as the data is stored in the same locations.

This instruction operates very much like the modification instruction:

9 XXX GO

is the general format, where XXX is the step number containing special information. This information should be in the format

SSAA +

where SS is the step number to be modified, and the AA stands for the initial value of the operand address.

In the program listed above the initializing information would be

0150 +

meaning set the address of step 01 to 50. This information, like the modification information, should not appear anywhere where it might be read by the interpreter. Most likely, it would appear in step 5 right after the modifier.

6. PAPER TAPE EQUIPMENT

In addition to the keyboard and the typewriter, the PDS 1020 Computer is equipped with a paper tape reader and paper tape punch for input and output purposes. The punch is used to store programs and data permanently on tape. The paper tape reader is used to enter such programs or data rapidly and accurately into the computer.

Paper Tape Reader

The paper tape reader is normally used in one of three ways. It may be used to enter programs for storage in memory; such programs normally consist of instructions and constant data but do not include variables. For example, assume that the program entered from tape includes a series of steps for solving $y = 2x$. Since 2 is a constant it may be entered along with the instructions and stored in memory permanently. The value of x is likely to change, however, and an INPUT VAR instruction is entered in its place. Once the program has been stored in memory and is in the process of being executed, the computer will come to the INPUT VAR instruction and halt; at that point the current value for x may be entered either from the keyboard or through the paper tape reader.

The second use of the paper tape reader is for entering data during execution. Using the same examples, let's assume that 100 values of x will be used by the program. These values may be input during the execution of the program from paper tape rather than entering each data value from the keyboard.

The PDS 1020 Computer has also a unique application for paper tape not usually found in stored-program computers. The paper tape reader may be used as a fast keyboard to enter instructions which are directly executed and data which is directly operated on. Thus instead of entering a program and storing it in memory and then executing it, the program can be read from paper tape and executed directly by the PDS 1020 Computer.

Paper Tape Punch

The paper tape punch is similarly used to prepare tape for the paper tape reader. The punch may be used to output the contents of the accumulator on paper tape. Thus the results of computations of one program may be punched out on paper tape and used as input for another program. The 100 y values computed by the program cited above, for example, could be punched out on paper tape and used by another program to compute Z where $Z = fy$.

One of the main applications of the paper tape punch is to provide permanent storage for use of programs retained in memory. A retained program stays in memory only until it is replaced by another program or until memory is cleared. Since with the PDS computer memory is automatically cleared every time power is turned off, retained programs should be punched out on paper tape if they are to be used repeatedly. Once the retained program is punched on paper tape it may then be reentered through the reader quickly and accurately.

Finally, in the PDS computer, the capability is provided to punch keyboard entries made during the step-by-step procedure, punching out operations and constant data values as a solution progresses. The tape arrived at in this manner can then be used as described previously to operate the computer and execute the program a second time, using the paper tape reader as a fast keyboard. It is again advisable to enter only constant

values into such a program and to leave the variables out. Before the tape is executed the user may enter and store his data values as he would if he were operating the computer through the keyboard.

Keyboard-Tape Combinations

These various modes of operation of the paper tape equipment may be combined to arrive at the most efficient and satisfactory problem-solving combination. For example, if the programs to be executed are very long and take up more than the memory space provided, it is possible to use the paper tape reader to enter instructions for the linear portions of the program, that is, the steps that progress sequentially from one operation to another, and to reserve the memory space for those operations which must be repeated in program loops or which require the logical decision-making capability of the computer.

Switch Settings

The computer control panel has four switches marked 1 through 4, located directly above the special function keys marked A through D. These switches are set ON by depressing them once and are turned OFF by depressing them a second time.

Reading Program

When switch number 1 is ON, the computer will accept instructions and data from the paper tape reader. If the program contains a RETAIN code, that is, if it is a retained program and previously punched out on paper tape, it will automatically be stored in memory. If the paper tape does not contain a RETAIN code it will be executed as it is entered with the computer handling each instruction as though it were receiving it from the keyboard.

Reading Data

When switch number 4 is turned on the computer will accept variable data from the paper tape reader during the execution of the stored program. The data is automatically entered into the accumulator and the stored program must then operate on it as desired by the user.

Needless to say, when reading tape the reader power switch must be turned on (see chapter on computer operating instructions).

Punching Accumulator

The contents of the accumulator may be punched on paper tape by using the instruction SUB+. This instruction is similar to the CPY+ instruction which types the contents of the accumulator on the typewriter. Like the CPY+ instruction, SUB+ may be used either during the step-by-step procedure or as part of the retained program. The contents of the accumulator is not erased by execution of this instruction.

Punching Retained Program

The instruction RETAIN N- will punch out the entire retained program number N. N must be a number from 1 to 6 and must refer to a program previously entered into memory in the RETAIN mode. The RETAIN command is punched on the paper tape along with the program itself, so that when the paper tape is read through the paper tape reader it will automatically be retained in memory. In common usage it is good practice to type out the program retained in memory before punching it on tape. Thus an EXECUTE N- instruction should be issued first to print out the retained program. Once the program has been typed out and checked by the user it may then be punched on paper tape for permanent use and storage.

During step-by-step operations, each keyboard entry can be punched on paper tape as the program is entered and executed, by turning on switch number 2. At the risk of over-stressing the obvious, the Punch power should be turned on before any punching instructions are executed.

7. MACHINE OPERATING PROCEDURE

Starting the Computer

In preceding chapters we have described how data and instructions may be entered into the computer through the keyboard, or through the paper tape reader, and how the computer is operated to solve problems and execute programs. So far, however, we have always assumed that the computer power has already been turned on and that the interpreter program itself had previously been entered into the computer. In this chapter we will show you how to start the computer, set the switches, load the paper tape reader and punch units, and how to get out of trouble if trouble develops.

To operate in the manner described in this guide the computer must first be loaded with the interpreter program itself. This program is supplied on a reel of tape and must be loaded through the paper tape reader. Remember that the PDS 1020 Computer clears all memory whenever power is turned off. When power is turned back on again the following procedure must be followed to load the interpreter:

1. Turn computer power switch ON.
2. Place the interpreter program tape in the paper tape reader as described below under the procedure for loading tape.
3. Turn reader switch power ON.
4. The START button light should be on by this time; depress the START button.

The interpreter will be read into the computer. When the entire tape has been read,

5. Depress reader EJECT button to feed through the tail leader.
6. Turn reader power switch OFF.
7. Depress START switch.

Rewind the tape and store for future use. The INPUT DATA light should now be on, signifying that the interpreter has been loaded and the computer is ready to accept data and instructions through the keyboard or through the paper tape reader.

Switch Settings

The four numbered switches may be set by the operator as follows:

SWITCH No. 1 ON -	the computer will read instructions and constant data from paper tape.
SWITCH No. 2 ON -	the computer will punch keyboard entries on paper tape.
SWITCH No. 3 ON -	the computer will type out the contents of the accumulator after each operation in the program, and return the typewriter carriage.
SWITCH No. 4 ON -	the computer will read data from paper tape during the execution of the program.

To enter a program from paper tape after the interpreter has been loaded, place the tape in the tape reader as described in the section on loading paper tape, turn reader power ON and depress switch number 1. Since the computer is in an idle state it will not start reading the tape until after it has executed an instruction. To start the tape reading process, enter any meaningless instruction from the keyboard such as DIV+ to return typewriter carriage. This instruction will return the typewriter carriage and the computer will then start reading the tape.

Trouble-Shooting

Human fallability being what it is, it is likely that some errors will creep

into the operation of the PDS 1020 Computer. Some of these errors can be detected by the computer and some cannot. If you meant to enter the number 12 for example and entered 21 instead, or if you meant to push the ADD button and hit the MPY button instead, you will simply get the wrong answer to your problem. The computer has no means of detecting such errors. On the other hand the computer can detect such errors as dividing by 0, taking the log of 0, an instruction to execute a program not in memory, and the like. When the computer detects an error it will halt and must be re-started manually. The procedure for re-starting will depend on the type of error discovered by the computer.

Display Lights

Note the group of 17 lights on top of the computer panel. These are register display lights which are used to display the contents of the various registers in the computer. Directly beneath the register display lights you will find the register display switch which can be set to any number from 0 through 9. When this switch is set to 6 the contents of the accumulator is displayed in the lights. The register lights are divided into four groups with four lights in each group marked 1, 2, 4 and 8 from right to left. In addition the 17th light is marked with a minus sign. When a light is on, it has the value of the digit inscribed on it; when a light is off its value is 0. When the minus light is on the contents of the accumulator is negative.

Error Stops

When the computer stops due to an error which it discovered, the number in the accumulator will often reveal the nature of the error and what steps should be taken to correct it. The descriptions below indicate the nature of the errors represented by given numbers in the accumulator.

Number in Accumulator	Error and Remedy
0000 0000 0000 0001	You entered an instruction or data value in the wrong form from the keyboard. For example, ADD 100 GO, 1., etc. Depress START and enter correct instruction or data.
0000 0000 0000 0020	You are trying to take the log of zero. Depress START, enter new instruction.
0000 0000 0000 0021	You are trying to divide by zero. Depress START and enter new instruction.
0000 0000 0001 0401	You tried to execute a program not in memory. Depress START, EXECUTE, the corrected program number, and the + key.
0000 0000 0000 8401	You have reached the end of scratchpad memory during your modification of an operand address (see Example 5). Depress START.
0000 0000 0000 8021	You have reached the limit of instructions which can be retained by the machine. Depress START to terminate the retain mode.
0000 0000 0000 8000	You have entered a wrong instruction in the retain mode. Depress START, enter correction.
0000 0000 0000 0421	You depressed RETAIN followed by a 0 or an 8. These are illegal entries. Depress START, the correct program number (1 - 6), and the + key.
	Note that RETAIN 7 has a special meaning, as

noted below, and RETAIN 9+ will be interpreted by the machine as RETAIN 1+.

0000 0000 0001 0420	You called for your program to be punched on tape but entered the wrong program number. Depress START followed by correct program number.
0000 0000 0000 8421	You called for your program to be typed out but used the wrong program number. Depress START, EXECUTE, the correct number and the GO key.

Step-by-Step Typeout

Errors which cannot be caught by the computer can sometimes be detected by typing out the contents of the accumulator after each operation in the retained program. To do this execute the program with SWITCH No. 3 ON. The contents of the accumulator will be typed out after each operation and the typewriter carriage will be returned automatically.

Error Correction

In the step-by-step procedure it is simple to rectify an error: you simply enter the correct instruction or the correct data value and ignore the result that was produced by the erroneous instruction or data value. To change an instruction in a retained program is likewise a simple matter. You must give the computer some information, however, regarding which instruction in which program it must change and what the new instruction is. For example, in program number 5, step 27, you entered an instruction MPY 10 GO; this instruction should have been DIV 10 GO. To change the instruction, the following procedure is necessary:

RETAIN 7+ initiates the correction procedure. This instruction tells the computer that you wish to change an instruction in a retained program. It is followed by the program number and the step number, each of them terminated with a plus, and then the correct instruction.

Thus the sequence

```
RETAIN 7+
5 +
27+
DIV 10 GO
```

would alter the 27th step of program number 5 to the corrected instruction DIV 10GO. If the wrong program number is given to the computer during this sequence, that is the number of a program not in memory, the computer will halt and the number 0000 0000 8000 0000 will be in the accumulator. Depress the START button and begin the procedure all over again.

Override

Occasionally the computer will get into an endless loop and will refuse to stop altogether. If this happens depress the OVERRIDE button, (the light will come on) release OVERRIDE by depressing the button again, enter through the keyboard 2000+. The INPUT DATA light will come on and the computer will be ready to accept new instructions.

Reset

If the computer halts for reasons other than these mentioned above, and will not accept instructions from the keyboard, use the following procedure as a remedy:

1. Depress the OVERRIDE button.

2. Depress the RESET button.
3. Depress START (the OVERRIDE light should come on).
4. Release OVERRIDE by depressing the button a second time.
5. Enter through the keyboard 2000+.

The INPUT DATA light will come on and you may proceed to enter instructions.

Keyboard Clear

Occasionally you may hit the key on the keyboard when the computer is not ready to accept a keyboard entry. In such a case the key will remain locked in the down position and may be released from this position by depressing the KEYBOARD CLEAR button at the lower right hand of the control panel.

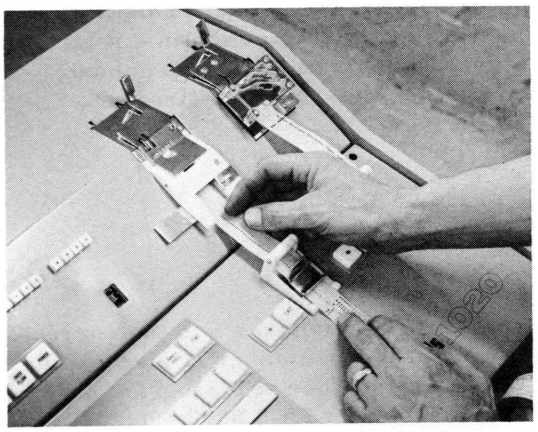
Error Clear

If the ERROR CLEAR light comes on any time during the operation, this is an indication of an error made by the computer. The light can be turned off by depressing the button. Then turn Power off, wait 30 seconds, turn Power on and re-enter the interpreter and your program. If the error light comes on again and persists, this may be an indication of a machine malfunction and the computer may have to be serviced before it can operate correctly again.

In addition to the indicators and keys described, you will find a key marked SINGLE CYCLE to the left of the START button and an indicator marked OVERFLOW. These have no meaning when operating in the interpretive mode, and are used during machine language operations only. They need not concern you here.

Loading the Paper Tape Reader

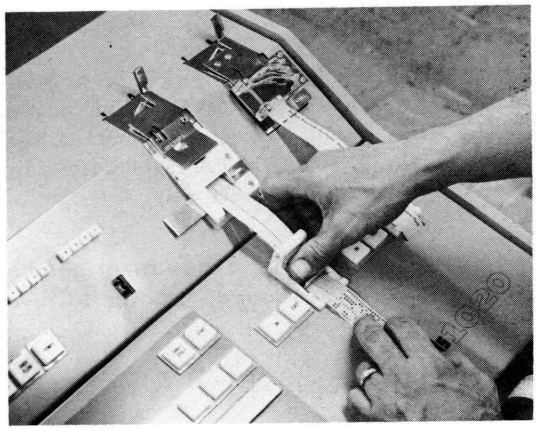
A



1. Open the spring-loaded switch and pull tape through.
2. Push tape into tape-guide as shown in Photo A.
3. Turn POWER switch ON.
4. Depress spring-loaded switch with your thumb as shown in Photo B. Reader will thread automatically.

When loading a program, with the interpreter already loaded, the following steps should be added:

B



5. Depress sense switch no. 1 to read tape.
6. Depress DIV +.
7. When tape is about half done, depress sense switch no. 1 a second time to release it.
8. When tape has been read, depress EJECT switch to release the tape.
9. Turn reader POWER switch off.

Loading the Paper Tape Punch

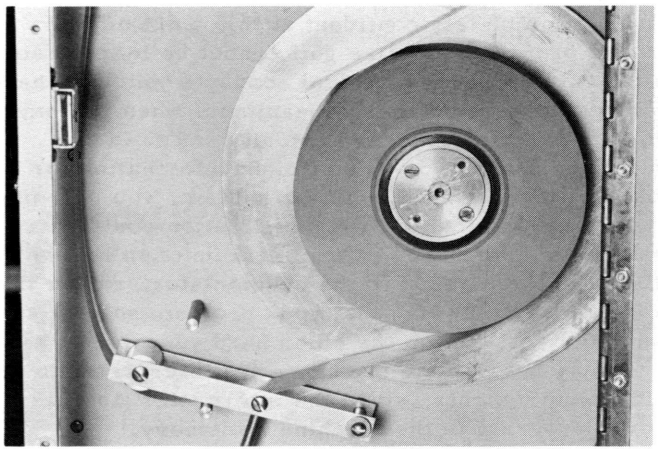


Photo C

1. Open service door on right side of computer.
2. Unscrew and remove front plate from paper tape reel.
3. Position tape on hub and pull a length of tape through pivot bar. Thread as shown in Photo C.
4. Replace front plate on reel.
5. Pull tape through opening to top of computer and close the service door.
6. Thread tape under spring-loaded switch, and through tape-guide.
7. Turn POWER switch ON.
8. Depress spring-loaded switch with your thumb.
9. Depress PUNCH FEED switch to thread tape and punch a length of leader as shown in Photo D.
10. To punch data or instructions use switches and commands described under Paper Tape Equipment.

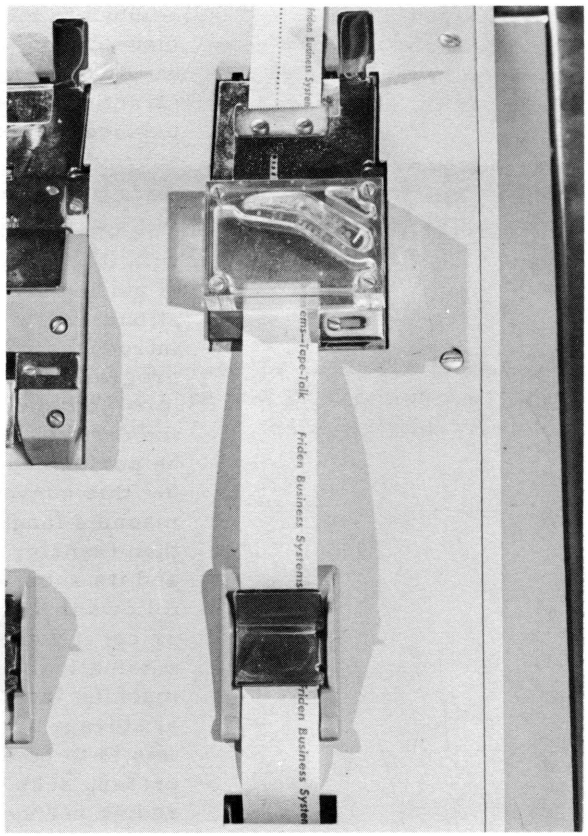


Photo D

8. AND WHERE DO WE GO FROM HERE?

You have now learned to operate the PDS 1020 Computer in the interpretive mode. You may not feel completely confident at this point of your abilities but remember that programming like golf cannot be learned simply by reading a book or a guide. The theoretical concepts which we have discussed in this guide can only become really meaningful when you have applied them to problems of your own and have actually sat down and solved them on the computer. You now have sufficient information for this purpose. When you get more familiar with the system, you may find that certain problems which are common in your application could be solved more conveniently if the computer had a special function key which would evaluate the factorial of the number in the accumulator, rather than a trigonometric function. Or you may find that your problem solving steps rarely run to more than 100 steps, but on the other hand you would like to store 200 data values. It may be possible to change the interpreter to include provisions for such requirements, since the interpreter itself is nothing more than a stored program in the machine's memory.

What Is An Interpreter?

The interpreter is a machine language program written with a special purpose in mind: instead of processing data the interpreter program accepts information from tape or keyboard or the special function switches; it interprets this information and determines its purpose and then transfers control to an appropriate sequence of machine language instructions which execute the operation called for by the interpretive command. The interpreter program must be entered into the computer and stored in memory before the computer can operate in the interpretive mode as described in this guide. This is the reason for the limiation on storage of instructions and data: the PDS 1020 is standardly equipped with 2048 words of memory, 450 of which you may use for storing interpretive instructions. 100 scratchpad locations are provided for data storage. The balance of memory is occupied by the interpreter program itself and its subroutines.

Interpreter Advantages

The interpretive mode of operation offers many advantages. To begin with it is very simple to learn and to use. It offers you a convenient way of getting acquainted with computer programming and with computer operations, while doing useful work and solving problems on the machine. It introduces you painlessly to some of the more sophisticated concepts of programming such as looping, address modification and the like. It allows you to enter numbers in scientific notation (floating point) and adjusts the decimal point automatically. In machine language this function must be performed by the programmer. There is a price however to be paid for this convenience and ease of operation: the interpreter is slower than machine language, since it must first interpret the command it receives then transfer to a subroutine to execute it. Furthermore, the interpreter and its subroutines take up a considerable amount of memory space. It follows that the capabilities of the interpreter are proportionate to this price; if you want the interpreter to evaluate the factorial of the number automatically for example, this can be easily done by adding a routine in machine language to do this. However, you will be trading some additional storage space for this capability and you will have to decide whether this is to be at the expense of another function performed by the interpreter, such as a trigonometric function, or whether this capability should reduce the number of program steps which you may use for solving your problems.

Added Capabilities

The capabilities of the interpreter can also be expanded from a hardware point of view. For example, you may add additional memory modules to the 2048 word capability of the computer. Additional memory may be divided, in any proportion you see fit, between additional data storage and additional storage for instructions. You may even want to vary this proportion from program to program and to set the limits on scratchpad locations and instruction storage yourself each time you enter a new program into memory. If your computer is equipped with the optional typewriter input capability, the interpreter can also be expanded to include alphanumeric capabilities: you may wish to type in headings which the computer will then type back out again as part of the solution of your problem.

The important point we are trying to make is that the interpreter is entirely flexible, since it is a program and not a piece of electronic hardware. Its capabilities and limitations can be expanded or contracted to suit individual applications.

What Next?

Once you learn to operate the PDS computer in the interpretive mode you have learned all the basic principles of programming a computer. As you grow more proficient with practice, you will be able to enter problems and solve them almost as fast as it would take you just to write the expression. If you use the computer extensively you may eventually want to master yet more sophisticated uses of the machine. You may wish to make your own modifications in the interpreter to suit your individual needs.

The PDS 1020 is a computer you can operate now. You can start solving problems on this machine as soon as it is installed. But more than that, the PDS 1020 Computer gives you the assurance that you will not outgrow its capabilities. It is a full-scale digital computer with a flexible and powerful wired-in command list, and may be used at any level of sophistication at which you want it to operate. When you are ready for the next step so is the computer.

The PDS Programmer's Handbook describes the machine language of the PDS 1020 and how it may be used. We recommend, however, that you become thoroughly familiar with the interpretive mode and actually operate the computer, using the interpreter, before you attempt to familiarize yourself with the machine language.

We believe that once you have learned to operate the PDS 1020 and when you fully appreciate the power and convenience that it places at your fingertips, you will use it as you use your slide rule: as a basic tool for solving engineering problems.

APPENDIX

Table of Printout Abbreviations
Summary of PDS 1020 Interpreter Instructions
Additional Examples

Table of Printout Abbreviations

PRINTOUT	COMMAND
Lxxx -	LDS XX GO
Cxxx -	CPY XX GO
Axxx -	ADD XX GO
Sxxx -	SUB XX GO
Mxxx -	MPY XX GO
Dxxx -	DIV XX GO
Dnnn + ...	TEST-JUMP N+
3007 -	INPUT VARIABLE
3008 -	SINE
3009 -	COSINE
300L -	SQUARE ROOT
300C -	EXP
300A -	ATN
300S -	LOG
30NM	EXECUTE N +
6000 -	RETAIN

SUMMARY OF PDS 1020 INTERPRETER INSTRUCTIONS

A stands for Accumulator. ( ) stands for the value contained in

	COMMAND	OPERATION
ARITHMETIC AND STORAGE	LDA XX GO	Load A with (XX)
	CPY XX GO	Copy (A) to xx.
	ADD XX GO	Add (XX) to (A). Result in A
	SUB XX GO	Subtract (XX) from (A). Result in A
	MPY XX GO	Multiply (XX) by (A). Result in A
	DIV XX GO	Divide (A) by (XX). Result in A
SPECIAL FUNCTIONS	SINE	Sine (A) in radians < 30,000.
	COSINE	Cosine (A) in radians < 30,000.
	SQUARE ROOT	$\sqrt{(A)}$ absolute value.
	ARCTANGENT	Arctangent (A).
	EXPONENTIAL	$e^{(A)}$. (A) 9.2.
	LOGARITHM	Log (A) absolute value. (A) $\leq$ 10,000.
RETAINED OPERATIONS *	RETAIN N +	Store program N. N = 1 - 6.
	RETAIN	End Retain mode.
	INPUT VARIABLE	Input data value from keyboard or tape.
	TEST/JUMP N +	If (A) negative, jump to N.
	7 XXX GO	Jump to XXX.
	8 XXX GO	Modify address per XXX.
	9 XXX GO	Initialize address per XXX.
	SSII	Modify or initialize step SS by or to II.
	EXECUTE N +	Execute program N.
TYPEWRITER FUNCTIONS	CPY +	Type (A).
	DIV +	Carriage return.
	EXECUTE N -	Type program N.
PUNCH FUNCTIONS	SUB +	Punch (A).
	RETAIN N -	Punch program N.

* Can be executed only in Retain mode.

ADDITIONAL EXAMPLES

The following two examples show some typical applications in providing quick answers to common engineering problems.

One example shows how you may compute the mean and standard deviation for a set of numbers. The other shows a table lookup function — using the initializing and modifying instructions to look up the appropriate data to be used in the calculation.

In the examples the following abbreviations are used:

L	=	LDA	C	=	CPY
A	=	ADD	S	=	SUB
M	=	MPY	D	=	DIV

Compute: $\mu = \frac{1}{N} \Sigma x_i$

$$\sigma = \left[\mu^2 - \frac{1}{N} \Sigma x_i^2 \right]^{1/2}$$

Input: N, x_i

	RETAIN 1+	
1.	INP VAR	Enter N
2.	C1 GO	Save
3.	D1 GO	
4.	C2 GO	1
5.	S2 GO	
6.	C4 GO	Zero
7.	C5 GO	
8.	L1 GO	N
9.	S2 GO	
10.	C8 GO	N - 1
11.	INP VAR	x_i
12.	C7 GO	
13.	A4 GO	
14.	C4 GO	Σx_i
15.	L7 GO	x_i
16.	M7 GO	x_i^2
17.	A5 GO	
18.	C5 GO	Σx_i^2
19.	L8 GO	Test
20.	S2 GO	If
21.	C8 GO	Done
22.	T/J 24 +	Yes
23.	711 GO	No
24.	EXECUTE 2+	To subroutine
25.	RETAIN	END

	RETAIN 2+	
1.	L5GO	Σx_i^2
2.	D1 GO	N
3.	C GO	
4.	L4 GO	Σx_i
5.	D1 GO	N
6.	C +	Print μ
7.	C10 GO	
8.	M10 GO	μ^2
9.	S GO	
10.	SQ ROOT	Form square root
11.	C +	Print σ
12.	D +	Carriage Return
13.	RETAIN	END

Compute: $$f(x) = f(x_i) + \frac{f(x_i) - f(x_{i-1})}{(x_i - x_{i-1})} (x - x_i)$$

$$x_{i-1} < x < x_i$$

Input: Table of Values:

2	x_1
3	$f(x_1)$
4	x_2
.	.
.	.
.	.
.	x_N
.	$f(x_N)$

X in accumulator on entry

Output: f(x) in accumulator on exit

RETAIN 3+

1.	CGO	Save Input	20.	L( ) GO	$f(x_i)$
2.	927 GO		21.	C98 GO	Save
3.	928 GO	Set	22.	S( ) GO	$f(x_{i-1})$
4.	929 GO	Table	23.	M97 GO	$(x-x_i)$
5.	930 GO	Entries	24.	D99 GO	$(x_i - x_{i-1})$
6.	931 GO		25.	A98 GO	$f(x_i)$
7.	LGO	X	26.	737 GO	Skip to END
8.	S( ) GO	X_i	27.	802 +	
9.	T/J 16 +		28.	1702 +	
10.	832 GO	29	29.	1800 +	
11.	833 GO	Modify	30.	2003 +	
12.	834 GO	Table	31.	2201 +	Step number
13.	835 GO	Entries	32.	802 +	and modifiers
14.	836 GO		33.	1702 +	
15.	77 GO	Try Again	34.	1802 +	
16.	C97 GO	$(x-x_i)$	35.	2002 +	
17.	L( ) GO	x_i	36.	2202 +	
18.	S( ) GO	x_{i-1}	37.	RETAIN	END
19.	C99 GO	$x_i - x_{i-1}$			