DIBOL for Beginners

Order No. AA-BI77A-TK
April 1984

Supersession:	This is a new manual.
Operating System:	VAX/VMS, CTS-300, RSTS/E, Professional, RSX-11M-Plus, Micro/RSX, Professional CTS-300
Software Version:	Applicable to all products containing DIBOL-83

First Printing, April 1984

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may only be used or copied in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by DIGITAL or its affiliated companies.

The specifications and drawings, herein, are the property of Digital Equipment Corporation and shall not be reproduced or copied or used in whole or in part as the basis for the manufacture or sale of items without written permission.

The following are trademarks of Digital Equipment Corporation:

CTI BUS
DEC
DECmate
DECsystem-10
DECSYSTEM-20
DECUS
DECwriter
DIBOL
digital
MASSBUS
PDP
P/OS
PRO/BASIC
Professional
PRO/FMS
PRO/RMS
PROSE
Rainbow
RSTS
RSX
Tool Kit
UNIBUS
VAX
VMS
VT
Work Processor

CONTENTS

CONTENTS (CONT.)

PREFACE

DIBOL For Beginners has been written for the professional who has had no previous experience in programming. The manual is an introduction to the fundamentals of programming and the vocabulary and syntax of DIBOL. After reading this manual, you will know the requirements for writing simple DIBOL programs and understand the concepts of data processing.

To increase your knowledge of DIBOL and to write more complex programs, refer to the publications *Introduction to DIBOL-83* and *DIBOL-83 Language Reference Manual*.

INTRODUCTION

A computer is a tool which you can use to solve business problems. Many people imagine that a computer has some type of "brain" that figures out what to do with a business operation and magically makes office life a lot easier. The real "brain" which makes a computer work successfully and efficiently in your business is yours. Before a computer can perform any of its daily "magic", you must decide what information is important to your business and how you will organize that information.

A computer actually makes your life easier by performing redundant tasks relatively quickly on large amounts of information. This is called "processing" and the information which the computer works with is referred to as "data".

You decide what data is to be stored in the computer and what type of processing will be performed on the data. The type of data that you select to be stored in the computer can be related to your daily business operations. Certain information about your customers, vendors, or inventory is used over and over again in daily transactions; for example, the customer name and address, a policy number, or an inventory code.

By examining the information already collected in your customer file folders or inventory stock sheets, you can begin to decide what pieces of data are necessary in order for you to run your business.

Once you have determined what data is necessary, you must decide on some consistent format or structure for storing the data. For example, if your business is customer oriented the data you store on each customer could look something like this:

1. Customer No_____
2. Name_____
3. Address_____
4. City_____
5. State__
6. Zip Code_____
7. Phone No_____
8. Contact Name_____
9. Salesman_____
10. Sales Code_____
11. Sales Hist_____

Of course there would be much more information than what appears here, but the format would be consistent for each customer. This consistency is the key to how a computer finds information so quickly. Since the data is organized in a certain way, the computer "knows" that certain data is in a certain location. Thus, when you request information on Mr. Jones, the computer doesn't really know how to tell Mr. Jones' record from Ms. Smith's record; it just compares what you've requested with all customer names until it finds one that matches. The computer can find names quickly because a customer's name is stored in the same place in each customer record. A computer can retrieve data so rapidly, it seems as though it "knows" what to do when in fact, it references information only by location.

A computer gives you the time to be more flexible and creative with the information you already have. Perhaps you've always wanted to have an end-of-week report which would give you a review of a particular aspect of your business. Unfortunately, you've always been so busy with daily tasks there is no time to review all of the customer or transaction records for this information. A computer will perform routine tasks more quickly thus giving you more time to produce special reports. You must plan in advance in order to capture this type of information. For example, if you wanted to generate a report on how many new accounts were opened on a particular day, part of your customer data format would have to include a place to store the date when the customer's account was opened.

A program tells your computer what to do with the data it is storing. A computer can calculate customer billing totals because it is instructed by a program to find amounts stored at certain locations and add these amounts together. Any processing your computer does is accomplished through programs which instruct the computer on how to do this processing. A computer can solve your business problems because special programs can be written to produce the results needed to solve your particular problems.

DIBOL is a computer programming language ideally suited for the solution of business problems. The letters in DIBOL are taken from the words "DIgital's Business Oriented Language". By learning to program in DIBOL you are not only learning how to control your business' data, you are learning how to communicate with your computer.

CHAPTER 1
COMMUNICATING WITH YOUR COMPUTER

In order to enter a program into your computer, there must be some form of communication between you and the computer. This communication is accomplished through the computer's hardware and software.

Hardware refers to the physical part of the computer. This includes the mechanical and electrical parts of the computer as well as the attached auxiliary equipment.

The principal piece of hardware is the Central Processing Unit (CPU) which acts as the computer's control center. Peripheral devices are auxiliary pieces of equipment which help the CPU. Peripherals are used for short- or long-term data storage, or as a means of communication.

The computer communicates with you through peripheral devices known as output devices. Terminals and printers are output devices. Data storage devices such as tapes, diskettes, and disks are considered output devices as well. You communicate with the computer using an input device, usually your terminal (tapes, diskettes, and disks can also be used as input devices).

Input and output devices provide a physical communication link between you and your computer. As information is entered into the computer, it is handled by the software.

Software refers to all of the written procedures and rules that control computer operations. When you write a DIBOL program, you describe the data that you want the computer to process and include a series of instructions that detail the operations to be performed on the data. Thus, you are writing software fitted to your specific needs.

There are programs which already exist within your computer system. These programs are necessary for fundamental tasks (such as data entry) and facilitate the creation of new programs. The editor, compiler, and linker are examples of programs which you will use when writing your own DIBOL programs.

A programming language is a human-oriented way of communicating with your computer. Just as your spoken language follows certain rules and conventions, a computer language is written to conform to rules. This manual will help you understand the language rules of DIBOL so that you can communicate with your computer.

CHAPTER 2
HOW DATA IS STORED

In order to understand how computers access and manipulate data, you must have an understanding of how this data (from the largest to the smallest unit) is stored.

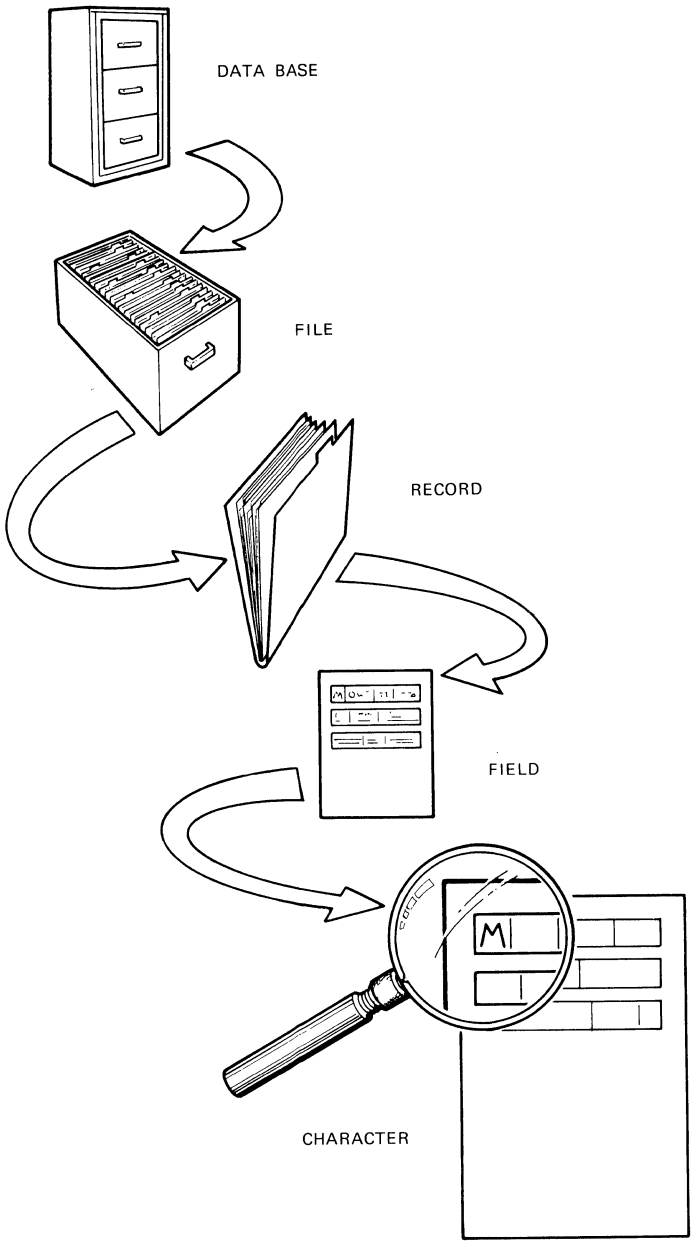


Figure 2-1 Data Hierarchy

Similar to the metal file cabinets which store the information relative to a business, the data entered into your computer is stored in a data base. This data base is made up of groups of files. For example, a manufacturing company with many employees would have a personnel file. This file would contain all of the data for all of the employees of the manufacturing company.

Each file is a collection of related records. A record could be a collection of all data items concerning one individual. Thus, the first record in the personnel file may be Mary Bara's record. In this record, Mary's social security number is recorded along with her name, address, job code, pay rate, etc. The second record would contain the same information relative to another employee.

Records are made up of fields. The data items mentioned above: social security number, name, address, etc., are the fields which make up the employee record.

Fields are made up of characters. A character is a letter, digit, or punctuation mark. Thus, Mary Bara's first name uses four characters in the first-name field.

By using a programming language, you can describe, create, and read files, and process the information contained in them.

Accessing Stored Data

Stored data falls into two major categories: data that must be accessed sequentially and data that can be accessed directly (this is also known as random access).

Sequential access means that data is obtained or stored relative to the location of the data which was most recently obtained or placed in storage. This means that the first 496 records must be read before record 497 can be located.

Direct or random access is the process of directly obtaining or storing data from a location. This means that record 497 can be located immediately and its data made available to the CPU.

The process of looking up a telephone number can be used to illustrate the difference between sequential and direct access. If you were using sequential access, you would locate Larry Zapp's telephone number by starting at page one in the telephone directory and sequentially comparing each person's name in the directory to Larry Zapp. If you were using direct access, you would go directly to the location in the telephone directory where Larry Zapp's telephone number is written and read the telephone number.

Direct access storage devices, such as disks, are the most popular way to store data. Each record on a direct access storage device can be accessed quickly without performing a lengthy sequential search. Thus if you are in a situation which requires that data be accessed and processed quickly, it is best to use a direct access storage device. If you are interested in long-term storage of data, a device which processes sequentially would be suitable.

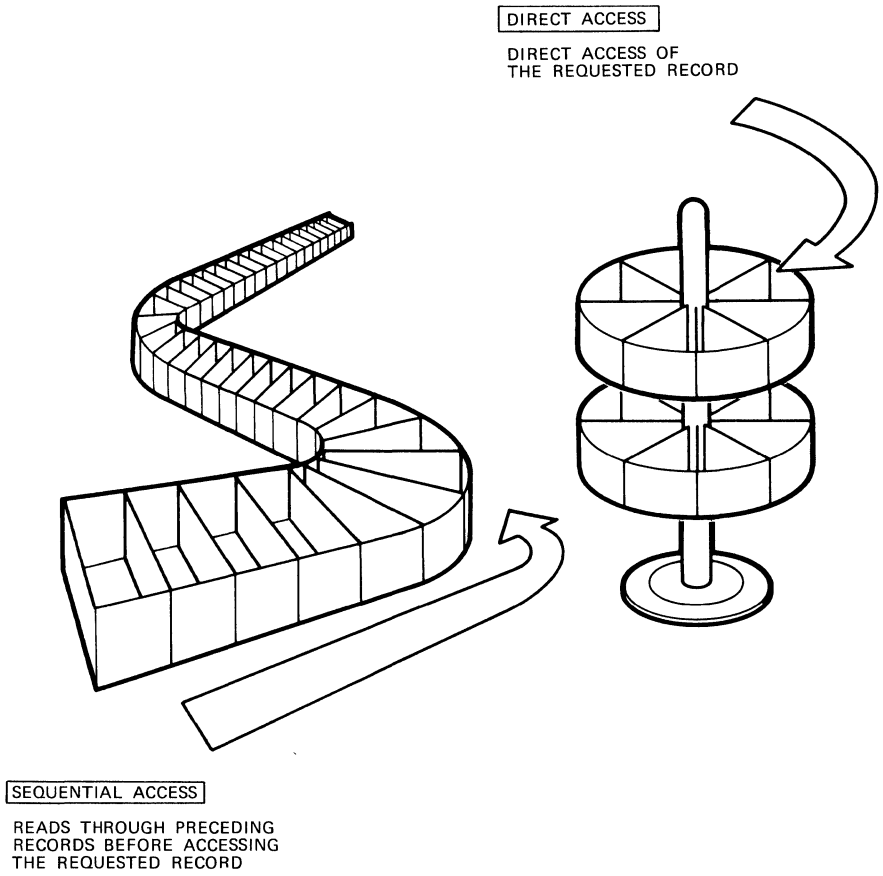


Figure 2-2 Types of Data Accessing

CHAPTER 3
HOW DATA IS PROCESSED

The Basic Data Processing Cycle

The basic data processing cycle is a standard sequence of events which exists in most programs. Fundamentally, the cycle consists of three events: input, processing, and output.

First, some type of information is entered or "input" so the program will have information or "data" with which to work. For example, a customer record is read from a file.

Next, the record is "processed". This means that calculations are made, some type of testing is done, formatting occurs, or information is extracted; the program instructs the computer on how the input is to be processed.

Finally, the results are "output". This output could occur in the form of a printed report, a display on a terminal screen, or revised data being placed back in storage. Often a combination of these forms of output takes place.

The Loop

In order for a sequence of instructions to be executed more than once, a program must contain a loop. This loop instructs the program to return to its beginning (or some other point) to obtain another record and repeat the steps of reading, processing, and outputing until a number of records have been processed.

The next figure illustrates a basic data processing cycle (including a loop). In this example, a record is read from a file and data from the record is formatted and printed. The program continues processing records until the end of the file is reached.

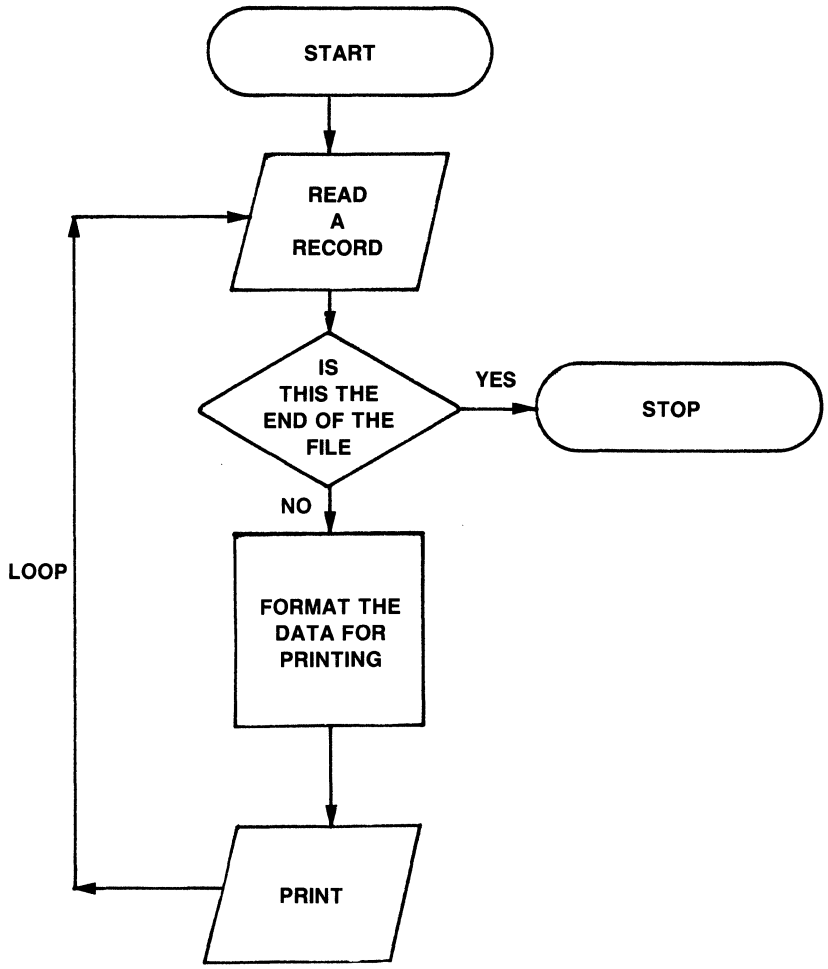


Figure 3-1 The Data Processing Cycle

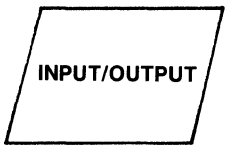
The Flowchart

In the preceding figure, the steps of the computer task are summarized and illustrated through a "flowchart". A flowchart is a diagram of the procedures the computer follows to solve a problem. A computer operates step-by-step; a flowchart enables you to see that you have not left out any logical steps in your program.

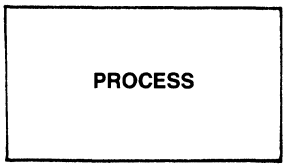
A standard set of symbols are used when developing a flowchart. A brief description of the action to occur at a particular step of the procedure is written inside of each symbol. These symbols are connected with a "flowline" to indicate the movement from one step to another.

The following symbols are used in Figure 3-1. For more flowchart symbols, refer to the Appendix.

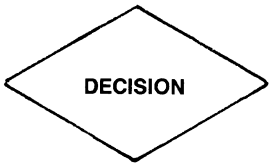
TERMINAL - An oval indicates where the program starts and stops.



INPUT/OUTPUT - A parallelogram indicates the input or output of data; that is, it represents information available for processing (input), or the recording of processed information (output).



PROCESS - A rectangle indicates that the computer is performing a processing task such as an arithmetic calculation or other defined operation causing a change in value, form, or location of information.



DECISION - A diamond indicates that a decision is made. For example, whether to go on with the following step or to "branch" to a different step.

FLOWLINE - An arrow indicates the direction the computer follows when it moves from one step to another.

Although there are several ways to diagram a program, flowcharting has been selected for this manual because it provides you with easy program visualization, it can be used immediately, and it is a programming standard that is not affiliated to any particular programming language.

CHAPTER 4
THE STEPS FOR WRITING A DIBOL PROGRAM

It is good practice to use the following nine steps when writing a DIBOL program:

1. Define the problem.
2. Flowchart the solution.
3. Code the solution.
4. Key in the program.
5. Compile the program.
6. Link the program.
7. Run the program.
8. Debug the program.
9. Document the program.

1. Define the problem

Before you can write an effective program, you must first define and analyze the problem. A well thought out solution to a business problem can save many valuable hours of programming time. It isn't unusual to find that the most "obvious" solution might be less efficient or unfeasible when working with a programming language. Once you have outlined the most effective approach, you can flowchart the solution.

2. Flowchart the solution

A flowchart is a diagram showing the sequence of program operations, or the steps to be taken to solve a given problem. By summarizing and illustrating the steps the computer will perform, you can observe problems in program logic prior to the actual writing of the program. In many cases, steps that were overlooked can be incorporated into the flowchart. The format of the results which will be output by the program can also be considered.

Flowcharting is only one of a variety of ways of visualizing your program. It is important to get a visual idea of how your program will operate whether you use diagramming or some abbreviated form of the language. The observation of program logic prior to the writing of the program is what proves to be beneficial.

3. Code the solution

You can now write or "code" the program using DIBOL. Your flowchart or diagram is a useful guide for translating your solution into DIBOL code.

4. Key in the program

Once a program has been coded, you then "key" the program into the computer using an editor. In most cases, this means you will type in your program at the computer terminal while checking for syntax and spelling errors on the terminal screen. An editor is a program included in your computer system that lets you create and modify text. (To learn more about the editor you will be using, reference your computer's user's manual.) At this time, it also beneficial to include "comments" within your program. These are notes which explain what the program is doing; they cannot be executed by the program. Comments will help you remember what the program is about when you or someone else modifies the program in the future.

At this stage, a program written in DIBOL code is called a "source program".

5. Compile the program

After the source program is keyed in, it must be compiled. Compilation is a process that converts DIBOL language to computer language. The DIBOL "compiler" in converting the source program to machine language, generates an "object file". At the same time that the compiler converts the program, it checks for proper DIBOL syntax. If errors are found, they are reported to you by the compiler. The compiler does not attempt to correct these errors; you must return to Step 4 to key in the corrections.

Although the object file is a translation of your source program, the computer still cannot run your program until the object file has been linked.

6. Link the program

The "linker" converts the object file into an "executable file" by associating it with other necessary program elements that are required of all DIBOL programs before they can be run. This executable file is the file that will actually be used by the computer. By "executing" the file, the program is run.

7. Run the program

The executable file is read into the computer's memory and the program that it represents is executed, producing the desired results.

8. Debug the program

If the program is producing incorrect results, you then have to "debug" or correct the problems in your program. "Bugs" can range from minor problems with printed headings to grossly inaccurate calculations and improper handling of input and output. Usually, debugging is a case of reviewing the program's logical sequence of steps and the usage of the DIBOL language within the program; either or both may be the cause of the problem or problems. The computer does only what it is told to do; if it is told to perform the wrong steps, the program will give the wrong answers.

Debugging requires you to return to the source program, make changes, recompile, relink and rerun your program. Do not be discouraged if the solution of one problem uncovers others.

9. Document the program

You should not consider a program completed until you have supplied program documentation. This includes instructions on how to run the program, a summary of assumptions, and any other information that will make future modification to the program as simple as possible. You should also return to the source program and type in comments which explain the program. When memory alone fails to remember what took place during program development, good documentation can come to the rescue.

Remember that these nine steps are a general guide to writing a DIBOL program. You will discover, as you gain programming experience, that there are exceptions, shortcuts, and even additional steps. Although these nine steps are applicable to all DIBOL programmers, review your user's manual for any steps which are unique to your system. In particular, reference your user's manual for instructions on using the editor and linker.

CHAPTER 5
DIBOL LANGUAGE SYNTAX

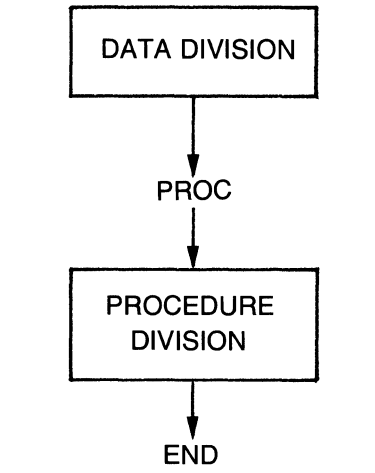
DIBOL Program Structure

A DIBOL program begins with the data division. This part of the program is used for "data declaration"; that is, it contains the statements and definitions which describe the data to be used in the program.

The word PROC (a DIBOL statement) follows the data division. PROC is used to separate the data division from the next part of the program, the procedure division. The PROC statement must be written in every DIBOL program.

The procedure division follows the PROC statement. This is where the actual program instructions are written.

The program is closed with an END statement.



DIBOL Program Structure

```
RECORD    ABC                          DATA DIVISION
          CUSTNM,  A10
          CUSBAL,  D6

          PROC

          OPEN (1,I,'CUSFIL.DDF')
          OPEN              (2,0,
          'CUSLIS.DDF')                PROCEDURE
                                       DIVISION
CKBAL,    READS (1,ABC, EOF)
          IF  (CUSBAL.GT.100)
          CALLOUT
          GOTO CKBAL
OUT,      WRITES (2,ABC)
          RETURN
EOF,      CLOSE 1
          CLOSE 2

          END
```

A DIBOL Program

The DIBOL Statement

A statement is an instruction to the computer to perform some operation. There are six types of DIBOL statements that can be used when writing a program:

1. Compiler Directives and Declaration Statements - These are statements which direct the compiler. A compiler instruction is not executable by the DIBOL program; it is understood only by the compiler.

2. Data Specification Statements - These are statements which describe the structure and type of data to be used in the program.

3. Data Manipulation Statements - These are statements which convert and assign values to data when the data is used in the program.

4. Control Statements - These are statements which control the order in which the steps of the program will be executed.

5. Intertask Communication Statements - These are statements which allow communication between programs.

6. Input/Output Statements - These are statements which control the transmission and reception of data between the program and the input/output devices.

Data specification statements are used in the data division of a program. Data manipulation, control, intertask communication, and input/output statements are used in the procedure division of a program.

A statement is made up of one or more elements. The first element is usually an English language verb that symbolizes an action to be performed on the data (such as OPEN, READ, WRITE, CLEAR, and CLOSE).

The other elements of a statement are called arguments. An argument is a value or an object on which the function (symbolized by the English language verb) will act. For example, in the statement CLEAR REC, REC is the name of the record that will be cleared; REC is the argument that is acted upon.

The following examples are complete DIBOL statements:

```
GOTO END

CALL TEST

IF (PAY.EQ.TOTAL) CALL PRINT
```

A program may contain only one statement per line. A statement can begin anywhere on a line and can contain up to 511 characters. Intervening, leading, or trailing spaces and tabs do not interfere with program execution and may be used to improve the readability of a program.

If you want to continue a statement onto the next program line, the ampersand symbol (&) must be the first character in the continued line. For example, you may write a brief statement:

```
SEC,        IF (INVENT .LT. ORDER) GOTO GETMOR
```

or you may need to use more room:

```
SEC,        IF (INVENT+ORDER .GT. SHIPPED
&           .AND.CASH.GT.MINIM) GOTO GETMOR
```

The ampersand symbol allows you to continue the statement onto additional lines.

Delimiters

Delimiters are the symbols which are used to separate the elements of the DIBOL language. The name and symbol used to represent each delimiter are listed in Table 5-1.

Table 5-1

DIBOL-83 Delimiters

Name	Symbol	Name	Symbol
Addition	+	Percent	%
Colon	:	Period	.
Comma	,	Pound	–
Division	/	Right Parenthesis	)
Double Quotes	''	Single Quote	'
Equal	=	Space	
Left Parenthesis	(	Subtraction	—
Multiplication	*	Tabs	< TAB >

The Statement Label

A statement label is a name that identifies a statement in the procedure division of a program. A label consists of a combination of letters and can also include digits. The first character of a label must always be a letter. A label can be up to six characters in length; any additional characters will be ignored. Thus, it is very important that the first six characters of every statement label be unique. A label may begin anywhere on a line, but must precede the statement it identifies and be separated from this statement with a comma. A label cannot be used to identify more than one statement. The following DIBOL statements have statement labels:

```
DET,    CLOSE 1

CHAIN, IF (INV.LT.ORDER) CALL MARK

LITE1,  OPEN (1,I,'VOLTS.DDF')
```

Normally, statements are executed by the computer in a sequential order. Labels can be used as arguments in procedure division statements in order to "alter" this program control. They allow you to bypass the sequential execution order and move to a specified statement.

In the following example, a label reference transfers program control to the statement having the label.

```
                GOTO TEST<------label reference
                .
                .
                .
label->TEST,    IF (PAY.EQ.TOTAL) CALL PRINT
                .
                .
                .
        PRINT,  WRITES (3,TOT)
                .
                .
                .
```

CHAPTER 6
A DIBOL PROGRAM

The easiest way to understand how a programming language works is to examine how it is used in an actual program.

Suppose you had a business and you wanted to know which customers owed a balance of more than $100. Provided the customer records were structured so they contained this information, you could write a program that would check the balance in each customer record for this specific condition. The program could then provide you with the names of the customers with balances exceeding $100.

The example program in this chapter sequentially reads through every customer record in an existing file named "CUSFIL.DDF". It checks each record to determine if the customer's balance is greater than $100. If the balance is greater than $100, the customer's name and balance are copied into a file named "CUSLIS.DDF". If the customer's balance is less than or equal to $100, the program continues on to read the next customer's record. The program continues in this manner until it reaches the end of the file.

Examine the example program:

```
.TITLE    'Customer Balance Program'

RECORD ABC
          CUSTNM, A10              ;Customer name
          CUSBAL, D6               ;Customer balance

PROC
          OPEN (1,I,'CUSFIL.DDF')  ;Open the input file
          OPEN (2,O,'CUSLIS.DDF')  ;Open the output file

CKBAL,    READS (1, ABC, EOF)           ;Read a record

          IF (CUSBAL .GT. 100) CALL OUT   ;If customer balance is
                                          ;greater than $100 call
                                          ;the OUT subroutine

          GOTO CKBAL       ;Read another record

          ;Subroutine to transfer the customer record to the
          ;CUSLIS.DDF file

OUT,      WRITES (2, ABC)
          RETURN

EOF,      CLOSE 1          ;Close the input file
          CLOSE 2          ;Close the output file

          END
```

Once you know where the data to be examined is located, in this case, the name and balance fields in the customer record, you can define this data in the data division of the program.

As illustrated in Figure 6-1, there is a direct correlation between the data in the customer record to be examined and the field definition statements in the data division of the program.

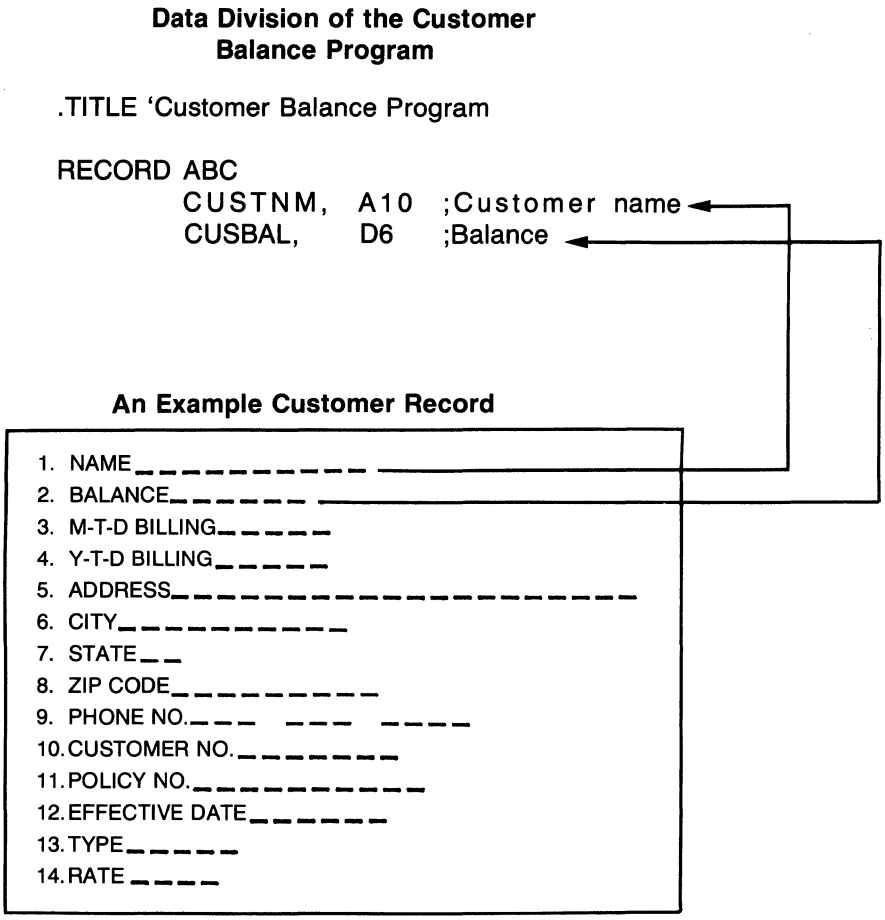


Figure 6-1 Customer Record/Data Division

The data to be used in the program is defined in the data division using data definition statements. The procedure division contains the instructions on what to do with the data. These instructions are written using various DIBOL procedure division statements. Once you are familiar with what each statement can do, you can write specific instructions for data which you have defined.

In the following paragraphs, each DIBOL statement used in the example program is explained and its function in the program is detailed. The statement descriptions are presented in the order in which they appear in the program. By reading these descriptions you will begin to learn the proper syntax of DIBOL statements which will enable you to write your own programs.

The .TITLE Statement

A DIBOL program can begin with the .TITLE statement. The .TITLE statement is used to place a heading on the top of every page of a DIBOL program listing. This is a helpful feature especially if the pages of your program listing become separated. .TITLE is followed by the text you want printed at the top of the page enclosed in quotation marks. This text is also known as an alpha literal. A literal is a value enclosed in apostrophe or quotation marks that does not change; a type of constant. Alpha indicates that the character is printable, whether it is a letter, number, or punctuation mark. A decimal literal is a number that will be used in calculation; it cannot be printed or displayed. In this example, "Customer Balance Program" is the alpha literal that will appear at the top of each page of the program listing.

THE DATA DIVISION

The data division is where the data to be processed is defined. All data divisions have at least one RECORD statement which contains a minimum of one field definition statement.

The RECORD Statement

The data division begins with the RECORD statement. This statement defines the area of memory where the data defined in the record will be stored. RECORD must be followed by at least one field definition. The total size of the fields within a record cannot exceed 16,383 characters. In the example program, RECORD ABC contains two field definitions (CUSTNM and CUSBAL) for a total size of 16 characters.

Record Name

Procedure division statements can reference all data defined in a record as a single unit when the record has a name. A record name consists of up to six characters, the first of which must be alphabetic. The remaining characters can be alphabetic, numeric, or the $ (dollars) or _ (underscore) symbols. Although you can enter a record name longer than six characters, only the first six characters of the name are significant; all remaining characters are ignored. A record name cannot be used to identify more than one record; all record names in a program must be unique. In the example, the record's name is ABC.

Field Definition Statement

The data in a record is defined using field definition statements. These statements let you name a data field, indicate what type of data is in the field, and specify its size. A field definition statement can also describe what is to be initially contained in the field.

All field definition statements are written in this format:

[name], $\left|\begin{matrix}A\\D\end{matrix}\right|$*n [,literal]*

where:

name	is the field name.
A	indicates the data in the field is alpha.
D	indicates the data in the field is decimal.
n	is the size of the field.
literal	is an optional alpha or decimal literal.

(Brackets indicate that an argument is optional while vertical lines mean a single choice must be made from the listed arguments.)

Field Name

In order for procedure division statements to easily reference a specific data field (without referencing the whole record), the field must have a name. A field name consists of up to six characters, the first of which must be alphabetic. The remaining characters can be alphabetic, numeric, or the $ (dollars) or _ (underscore) symbols. Although you can enter a field name longer than six characters, only the first six characters are significant; all the remaining characters are ignored. A field name cannot be used to identify more than one field; all field names in a program must be unique.

Data Type and Size

The data type specification for a field can be A for alpha data or D for decimal data. This is followed by the data field size; this must be a positive number that specifies the number of characters that the field may contain. The field size of an alpha field cannot exceed 16,383 characters. The field size of a decimal field cannot exceed 18 characters. For example:

CUSTNM, A10	Specifies an alpha field of ten characters.
CUSBAL, D6	Specifies a decimal field of six characters.

Literal

A literal lets you specify the initial value in an alpha or decimal field which you have defined. An initial value is specified for alpha data using apostrophe characters; an initial decimal value does not require apostrophe characters. Initial values are optional in data field specification statements. The value that you assign will be in a field when it is initially referenced by the program. Remember that this initial value can be changed by operations performed on it when the program is executed. If no value is specified, a default value is automatically entered during compilation. A default value is an assumption made by the computer or program when no specific value is given. If no literal is specified, an alpha field will initially be filled with spaces while a decimal field will initially be filled with zeros. The literal is inserted after the data type and size specification and must be preceded by a comma.

A literal must be the same data type and contain the same number of characters as specified for the field. Leading and trailing signs (+ and -) in decimal literals are not counted when calculating the size of a literal. Alpha literals must be enclosed in apostrophe (') or quotation (") characters.

The following example statements contain decimal literals:

1. NUM, D11, -99234780113

2. LIMIT, D3, 060

3. TAG, D6, +000431

Decimal literals contain only the plus (+) sign, the minus (-) sign, and the digits 0 through 9. Thus, "$1,000,000" must be designated as an alpha literal due to the dollar sign ($) and commas.

The following example statements contain alpha literals:

1. MSG, A19, "SALARY DATA INVALID"

2. DIV, A3, '$10'

3. LIN, A14, 'Payroll Number'

4. , A4, '****'

In the example program, two data fields are defined in RECORD ABC.

The first field, which will store the customer name, has the field name "CUSTNM". This is followed by a required comma and the field's type and size. The "A" stands for alpha data; the "10" indicates that the field is ten characters long.

The second field, which will store the customer balance, has the field name "CUSBAL". This is followed by a required comma and the field type "D" for decimal data. The "6" indicates that the field is six characters long.

Comments

To the right of the source code in the example program, you will see short descriptive notes relative to each line of code. "Comments" of this type can be placed on any statement line by preceding them with a semicolon (;). A program line that begins with a semicolon contains comments only. Although comments are optional, it is strongly recommended that you use comments as a means of documenting your program internally. They will prove useful in the future when you (or perhaps someone who is unfamiliar with the program's logic) must return to the source code to make some changes but have forgotten some of the program's original logic.

The Procedure Division

The statements in the procedure division define the logical sequence of the data processing cycle. For example, a file (or files) must first be opened before the program can assess the data required for the program operations. Once the data has been input, the program can perform various operations on it and then the results can be channeled to you in some form of output; a printed report, a terminal display, etc. All of the statements in the procedure division of a program reflect a sequence of events. The first statement in the procedure division of the example program is the OPEN statement, to open a file. This is followed by a READS statement, to read the data from a file, and the logic continues.

The PROC and END Statements

The procedure division is always separated from the data division by the PROC statement. This statement indicates to the compiler that the procedure division is beginning. The PROC statement is required in every DIBOL program. A matching END statement appears at the end of the program to indicate the end of the procedure division.

The OPEN Statement

In the data division, you define the type of data that will be used in the program. In order to reach the actual data that is residing in an existing file, you must first open the file containing that data thus making the data available to the program. The OPEN statement lets you open a certain file for a specific operation.

A basic OPEN statement follows this format:

OPEN (*channel*, *mode*, *filespec*)

where:

channel is a channel number.

mode is the method in which data will be transferred.

filespec is the file specification.

Channel

Because DIBOL allows you to have more than one file open at a time, you must indicate which channel to associate with a file so that records from one file do not get mixed up with records from other open files. A channel is similar to a telephone line. Each telephone conversation needs a separate telephone line, but many lines can be used at the same time. DIBOL allows a maximum of 15 channels to be open at once. You indicate a channel for your file with a number (1 through 15). Your program is associated with the file via this channel as long as the file is open.

Mode

Mode indicates how data will be transferred. There are three fundamental modes (input, output, and update) which are specified with the letters I, O, and U.

Input (I) is used to get data from an existing file and use it as input to the program. Data can only be read from a file using the input mode.

Output (O) is used to create a file. Data is written to a file using the output mode.

Update (U) is used to input and output data to records in an existing file.

Additional modes are available depending on the type of computer system you are using. Reference your system's DIBOL user's guide for additional capabilities.

File Specification

The file specification is used to specify the name of the file associated with the channel. The file name is enclosed in apostrophe or quotation marks.

File Extension

A file extension follows a file name and is used to indicate the file type. File extensions are usually three characters long and are immediately preceeded by a period. The system you use has its own extensions which it will automatically assign.

When you write a program, you assign the .DBL extension to the program name. The functions you perform on that program may create new files with different extensions. For example, when the program is compiled and the object file is output by the compiler, the extension will be .OBJ. When the file is linked, the linker will output an executable file with the extension .EXE (or .TSK). Remember, extensions are system dependent so refer to your system's DIBOL user's guide for the type of file extensions the system uses.

In the example program, two OPEN statements are used.

```
OPEN (1,I,'CUSFIL.DDF')

OPEN (2,O,'CUSLIS.DDF')
```

The first statement opens the file named CUSFIL.DDF and associates it with channel 1. Data will be input from this file to the program.

The second statement opens a new file named CUSLIS.DDF and associates it with channel 2. Data from the program will be output to create this file.

The OPEN statement examples in this manual show only a few of the OPEN statement's capabilities. To find out more about the capabilities of the OPEN statement, reference the *Introduction to DIBOL-83* and the *DIBOL-83 Language Reference Manual*.

The READS statement

Once a file has been opened, data to be used in the program can be read from the file. The READS statement inputs a record from a file. The first execution of this statement reads the first record in the file. Each time the statement is executed the next (successive) record is read.

The READS statement is written in this format:

READS (*ch*, *record*, *label)*

where:

ch is the channel number associated with the file from which you will read data. This channel number must be specified in an OPEN statement prior to its use in the READS statement.

record is the name of the alpha field or record which will contain the data.

label is the statement label where program control will be transferred when the end of the file has been reached.

In the example program, the following READS statement is used:

```
READS (1, ABC, EOF)
```

This statement indicates that record ABC will be read from the file associated with channel 1 (in this case, the file named CUSFIL.DDF). Every time the READS statement is executed a record will be read and the data placed in record ABC. When the end of the file is reached, the program control will be transferred to the statement label EOF.

The IF statement

In the example program, records are read and the customer balance field (CUSBAL) of each record is checked for a value greater than $100. If the customer balance is greater than $100, a certain function is performed. Thus, the course of action this program takes is dependent upon the value that is stored in the balance field for each record.

The IF statement is used to determine whether a certain condition is true. If the condition is true, a certain function is performed. The IF statement is written in this format:

IF *condition statement*

where:

condition is the expression which determines whether or not the statement is executed. If the condition is true, statement is executed. If the condition is false, statement is not executed and execution continues with the next statement in the program.

statement is a DIBOL procedure division statement. This statement specifies what will be performed if the condition is true.

In the example program, the following IF statement is used:

```
IF (CUSBAL .GT. 100) CALL OUT
```

The condition in this IF statement is "CUSBAL .GT. 100". (Enclosing the condition in parenthesis is optional, but is recommended as it makes the whole statement easier to read.) The value in the field named CUSBAL is compared with 100. If the value is greater than 100, the condition is true and the statement "CALL OUT" is executed. If the condition is not true, the program executes to the next statement in the program (GOTO CKBAL).

Relational Expression

The condition "CUSBAL .GT. 100" is a relational expression. A relational expression is a comparison of data using relational operators. The following relational operators are used in DIBOL:

.EQ. Equal

.NE. Not equal

.GT. Greater than

.LT. Less than

.GE. Greater than or equal

.LE. Less than or equal

When using relational operators, only like data types can be compared. Thus, decimal data must be compared to decimal data and alpha data must be compared to alpha data.

The following relational expressions are correct; like data is compared:

```
1. IF ('ABC' .EQ. 'ABE') STOP

2. IF (15 .GT. 10) RETURN
```

The following relational expression is incorrect; alpha data is compared to decimal data:

```
IF ('15' .EQ. 23) CALL INV
```

When evaluating an alpha relational expression, data is compared on a character for character basis from left to right and is limited to the size of the shorter piece of data.

For example, when comparing a three-character alpha field to a five-character alpha field, only the three leftmost characters of each field are used. Thus, the result of the following statement would be TRUE:

```
IF ('ABC' .EQ. 'ABCDE') STOP
```

The CALL Statement

The CALL statement transfers program control to an internal subroutine. A subroutine is a sequence of procedure division statements which perform a particular function. Because it is a "self-contained" routine, a subroutine can be written once and used at more than one point in a program. An "internal" subroutine is within a program whereas an "external" subroutine exists outside of the program. (For more information on external subroutines, refer to the XCALL statement in your *DIBOL-83 Language Reference Manual*.)

A CALL statement must be matched by a RETURN statement. The matching RETURN statement causes program control to return to the statement following the CALL statement.

The CALL statement is written in this format:

CALL *label*

where:

label is the statement label of the first statement in the subroutine.

Refer to the IF statement in the example program. In this statement, "CALL OUT" is used. This transfers program control to the subroutine that begins with the statement label "OUT". The subroutine ends with the RETURN statement. This returns program control to the statement which logically follows CALL OUT. In this case, it would be returned to "GOTO CKBAL".

The GOTO Statement

The GOTO statement transfers program control to a statement with a specified label. The GOTO statement is written in this format:

GOTO *label*

where:

label is the statement label of the statement where control is to be transferred.

In the example program, "GOTO CKBAL" instructs the program to go to the statement which begins with the statement label "CKBAL". Thus, the program goes back to the READS statement and reads the next record in the file.

The WRITES Statement

The subroutine in the example program transfers customer records to a file named CUSLIS.DDF, provided the customers have a balance greater than $100. To output these records to a new file, you must use the WRITES statement.

The WRITES statement transfers the data contained in a record, an alpha field, or an alpha literal to the next available space in a specified file. This is also known as sequential output.

The WRITES statement is written in this format:

```
WRITES (ch,record)
```

where:

ch is the channel number associated with the file to which you are writing data.

record is the name of the record, alpha field, or alpha literal which contains the data to be written.

In the example program, the following WRITES statement is used:

```
WRITES (2, ABC)
```

This statement indicates that record ABC will be written to the file associated with channel 2 (in this case, the file named CUSLIS.DDF). Every time the WRITES statement is executed, a new record will be written to the file.

The CLOSE Statement

When all program processing is completed, files that were opened with the OPEN statement must be closed. The CLOSE statement completes all pending data transfer to the file being closed and disassociates the channel number from the file. All open channels should be closed. It is possible for data to be lost or corrupted if a channel is not closed before a program terminates.

The CLOSE statement is written in this format:

```
CLOSE ch
```

where:

ch is the channel number associated with the file to be closed. This channel number must be specified in an OPEN statement prior to its use in the CLOSE statement.

A single CLOSE statement can close only one channel, therefore, a separate CLOSE statement is required for each channel to be closed.

In the example program, two CLOSE statements are used:

```
CLOSE 1
CLOSE 2
```

The first CLOSE statement closes channel 1 which is associated with the file named CUSFIL.DDF. The second CLOSE statement closes channel 2 which is associated with the file named CUSLIS.DDF.

CHAPTER 7
LOOPS

A computer would be of little help to you if it couldn't do repetitive work. DIBOL program loops enable a computer to repeat certain program statements. The instructions that control a loop let you specify under what conditions or how many times the loop will repeat. By using these instructions, which can take only a few program lines, you avoid writing separate instructions each time you wish to repeat a program operation.

The GOTO statement can be used to create a loop, as seen in the example program in Chapter 6. But because loops are used so often, DIBOL provides special instructions for them.

One way of repeating a set of instructions for a given number of times is to use a counter and the INCR statement. Examine the following program:

```
.TITLE    'Customer Balance Program'

RECORD ABC
       CUSTNM,     A10
       CUSBAL,     D6

RECORD
       CNT,        D3

PROC
       OPEN (1,I,'CUSFIL.DDF')          ;Open the file
       OPEN (2,0,'CUSLIS.DDF')          ;Open the file

       CNT = 0                          ;Set counter to zero

CKBAL, READS (1,ABC,EOF)                ;Read record

       INCR CNT          ;Add 1 to the counter

       IF (CUSBAL .GT. 100) CALL OUT

       IF (CNT .LT. 10) GOTO CKBAL ; Loop

EOF,   CLOSE 1       ;Close the file
       CLOSE 2       ;Close the file

OUT,   WRITES (2, ABC)
       RETURN

       END
```

In this program, the first ten customer records will be read from the file named CUSFIL.DDF. The value in the balance field of each of these records will be checked and if it is greater then 100, the record will be copied into the file named CUSLIS.DDF. When the ten records have been read, the program ends.

The field named CNT is the "counter" field. This field will keep track of how many times the loop has been repeated. The counter is assigned the value of zero when the program begins, using the statement CNT = 0. After a record is read, the INCR CNT statement is executed. This adds 1 to the value in the decimal field named CNT. INCR statements are written in this format:

INCR *dfield*

where:

dfield is the decimal field to be incremented.

The statement IF CNT .LT. 10 GOTO CKBAL controls the number of times the loop is repeated. The statement checks the value in the counter field. If the value is less than 10, program control returns to the statement labeled CKBAL and another record is read. If the value in the counter field is 10, the loop has been repeated the required number of times and the next statement in the program is read.

The CNT field contains a value which changes; this is called a variable. A variable is an entity whose value can change. Another type of value which can be assigned to a field is a constant. A constant is a value which does not change.

The FOR Statement

Another way to repetitively execute a statement is to use the FOR statement. The following format is used to write a basic FOR statement:

FOR *dfield* FROM *initial* THRU *final statement*

where:

dfield is the decimal field to be incremented.

initial is the initial value to be assigned to dfield.

final is the final value for dfield.

statement is a DIBOL procedure division statement.

Examine the FOR statement in the following example:

```
.TITLE          'Customer Balance Program'

RECORD          ABC
                CUSTNM,     A10     ;Customer name
                CUSBAL,     D6      ;Customer balance

RECORD
                CNT,        D3

        PROC

        OPEN (1,I,'CUSFIL.DDF')                 ;Open the file
        OPEN (2,O,'CUSLIS.DDF')                 ;Open the file

        FOR CNT FROM 1 to 10            ;As long as the counter contains
        BEGIN                           ;a value from 1 thru 10,
        READS (1,ABC,EOF)               ;read a record and
        IF CUSBAL .GT. 100 CALL OUT     ;check the value in the
        END                             ;CUSBAL field

EOF,    CLOSE 1
        CLOSE 2

OUT,    WRITES (2, ABC)
        RETURN

        END
```

FOR CNT 1 THRU 10 is the beginning of the FOR statement. This sets the initial value of the counter to 1. The counter is automatically incremented every time a record is read and its balance field checked. As long as the counter contains a value from 1 thru 10, the statement specified within the FOR statement will be executed. When 10 records have been read, the final counter value will have been reached and the loop will no longer be executed. The next procedure division statement will then be read.

Since the performance of two steps is desired in the example program (the reading of a record and the checking of its balance field), a BEGIN-END block is used. When these words are used together, everything that falls between them is treated as one DIBOL statement.

The Value Assignment Statement

The statement CNT = 0 is a value assignment statement. These statements are written in the following format:

destination = *source*

where

destination is a record or field name where the data is to be stored.

source contains the data that is going to be stored in the destination.

The contents of source is always moved to destination. The destination must be defined in the data division and can be either alpha or decimal data type. The source data is always converted to the data type defined for the destination.

The size of the destination field must accommodate the size of the source data. If the source data is too large for the destination, extra characters will be truncated (dropped off) and the data placed in the destination field will be inaccurate. If the source data is smaller than the destination, data is justified. This means that the data is positioned to the right or left of the field and the extra character positions are filled with zeros or spaces. Because each data type field has different rules for justification, you must pay special attention to the movement of data from one field to another.

Moving Alpha Data into an Alpha Field

When alpha source data is moved into an alpha type destination field, the data is left-justified. This means that all of the characters are moved to the left.

In the following examples, MSG is an alpha field which is six characters long.

```
MSG = 'HELLO'    ;MSG contains 'HELLO '
```

When HELLO is moved into the MSG field, the five characters of HELLO are left-justified in the field. Any remaining postions in the field are filled with spaces. Thus, the MSG field contains HELLO followed by one space.

```
MSG = 'WELCOME'      ;MSG contains 'WELCOM'
```

When WELCOME is moved into the MSG field, the seven characters cannot fit into the field. WELCOME is left-justified and the extra right-most characters are truncated. Thus, the MSG field contains WELCOM.

Moving Decimal Data into a Decimal Field

When decimal source data is moved into a decimal type destination field, the data is right-justified. This means that all of the characters are moved to the right.

In the following examples, COS is a decimal field which is four characters long.

```
COS = 195          ;COS contains 0195
```

When 195 is moved into the COS field, the three characters are right-justified in the field. Any unused field postion is filled with zeros. Thus, the COS field contains 0195.

```
COS = 19586        ;COS contains 9586
```

When 19586 is moved into the COS field, the five characters cannot fit into the field. 19586 is right-justified and the extra left-most characters are truncated. Thus, the COS field contains 9586.

Moving Alpha Data into a Decimal Field

When alpha source data is moved into a decimal type destination field, the data is right-justified. This means that all of the characters are moved to the right. The alpha source data may consist of up to 18 numeric characters and can include plus (+) and minus (-) signs. Spaces in alpha source data are ignored by the decimal destination field.

In the following example, ADF is a decimal field which is three characters long.

```
ADF = '12'         ;ADF contains 012
```

When 12 is moved into the ADF field, it is right-justified. Any unused position in the decimal destination field is filled with zeros. Thus, the ADF field contains 012.

```
ADF = '1234'       ;ADF contains 234
```

When 1234 is moved into the ADF field, it is right-justified. Since there is not enough room for all of the characters, the left-most characters causing the overflow are truncated. Thus, the ADF field contains 234.

If the source data contains any character that is not a number, a plus sign, a minus sign, or a space, the attempt to move it into the decimal destination field will result in the error message BAD DIGIT when the program is executed.

Moving Decimal Data into an Alpha Field

When decimal source data is moved into an alpha type field, the data is right-justified. This means that all of the characters are moved to the right.

In the following examples, BAC is an alpha field which is four characters long.

```
BAC = 123          ;BAC contains ' 123'
```

When 123 is moved into the BAC field, it is right-justified. Any unused position in the alpha destination field is filled with spaces. Thus the BAC field contains a space in the leftmost position, followed by 123.

```
BAC = 123456       ;BAC contains '3456'
```

When 123456 is moved into the BAC field, it is right-justified. Since there is not enough room for all of the characters, the leftmost characters causing overflow are truncated. Thus, the BAC field contains 3456.

```
BAC = 000          ;BAC contains ' 0'
```

If the source data has a value of zero, the destination field will contain a single right-justified zero with remaining character positions to the left filled with spaces.

CHAPTER 8
DIBOL MATH

Arithmetic can be performed in a DIBOL program using the following operators:

+	addition	8 + 2 = 10
-	subtraction	8 - 2 = 6
*	multiplication	8 * 2 = 16
/	division	8 / 2 = 4

Precedence of Operations

DIBOL processes an arithmetic expression according to a particular precedence. For example, multiplication is done before addition. The result of 4 + 5 * 7 is 39.

The order of precedence (from first to last) is:

1.	( )	parentheses
2.	+ and -	unary plus and unary minus (a signed number, i.e. positive or negative)
3.	#	rounding
4.	* and /	multiplication and division
5.	+ and -	addition and subtraction

DIBOL reads an expression from left to right and performs the operations according to the above order of precedence. If the operations are the same or on the same level, they are performed left to right. Multiplication and division are on the same level, as are addition and subtraction.

Changing Precedence

You can change the arithmetic precedence by using parentheses. DIBOL processes all operations within parentheses before anything outside the parentheses. The normal rules of precedence apply to all operations grouped inside of the parentheses. You can control the order of operations by grouping operations you wish to occur first within the parentheses. The parentheses must be matched. That is, there must be a closing parentheses for every opening parentheses.

Notice the difference in results in the following two examples:

Expression	Result
2 + 3 * 7	23
(2 + 3) * 7	35

Nested Parentheses

Parentheses that are within parentheses are called nested parentheses. You can use this other set of parentheses (inside the first set) to further control the order of operations. In the following example, the operation within the inner parentheses is processed first, then the operations within the outer parentheses, then the rest of the expression.

(2+7*(6+4))/9+2 The result is 10

More on DIBOL Math

DIBOL math deals only with integers. This means that decimal points, implied or otherwise, are not input to or output from a calculation. When dividing, the integer part of the quotient is retained while the remainder is discarded. Thus, 5/2 is 2 (not 2.5) and 1/3 is 0 (not .33). However, special arithmetic operations can be performed to avoid these limitations and if the results are to be output to a terminal or printer, they can be formatted so that information will be displayed or printed in any manner you want. Remainders can be retained and decimal points can be placed in the correct position. For information on the special arithmetic operations and formatting instructions refer to manuals such as the *Introduction to DIBOL-83*, the *DIBOL-83 Language Reference Manual*, or your system's DIBOL user's guide.

Examples

The following examples assume the data division of the program contains the following information:

```
RECORD
        MONEY,    D6,    127654
        Y,        D3,    -326
        A,        D1,    4
        B,        D2,    10
        C,        D2,    20
        D,        D1,    5

PROC
```

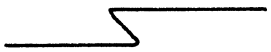
The following examples illustrate the use of arithmetic operators:

Expression	Result
A + B-C	-6
A*D	20
C/D	4
B/A	2 (The remainder is discarded)

The order of precedence can be modified by using parentheses, as in the following examples:

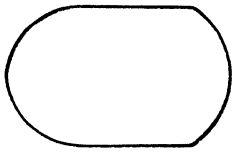
Expression	Result
B + C/D*A	26
B + C/(D*A)	11
(B + C)/(D*A)	1 (The remainder is discarded)
((B + C)/D)*A	24

APPENDIX
FLOWCHART SYMBOLS

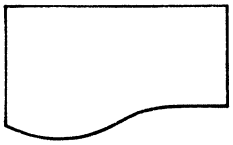


COMMUNICATION LINK
This symbol represents the transmission or reception of information by a telecommunication link.

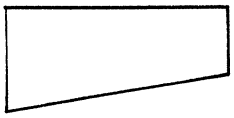
CONNECTOR
This symbol represents an exit to, or entry from, another part of the flowchart.



DISPLAY
This symbol represents information displayed by terminals, printers, plotters, etc.

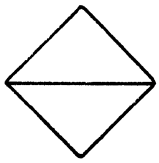


DOCUMENT
This is an output symbol which represents information to be printed.



MANUAL INPUT
This symbol represents information input from keyboards or other sources.

PREDEFINED PROCESS
This symbol represents one or more named operations or program steps specified in a subroutine or another set of flowcharts.



SORT
This symbol represents the arranging of a set of items into a particular sequence.

GLOSSARY

Alpha
A printable character; a letter, number, or punctuation mark.

Argument
The value or object upon which a function acts. In the statement CLEAR REC, REC is the name of the record that will be cleared; REC is the argument.

Branch
A selection between two or more possible courses of action in a program's flow of control, usually based on some condition.

Central Processing Unit (CPU)
The central piece of a computer's hardware which acts as the computer's control center.

Channel
A number associated with an open file that identifies the path by which data flows between a file and a program.

Character
A letter, digit, or punctuation mark.

Comments
Text that explains or identifies a particular part of a program step but has no effect on the computer's execution of program instructions. Comments inserted in a program help future users understand the program.

Computer
A device capable of accepting information, processing it, and providing results. A computer usually consists of a central processing unit (CPU), input and output devices, and storage units.

Constant
A value that does not change during the execution of a program.

Counter
A record or field used for storing a value that is increased or decreased as directed by program instructions. A counter is often used to control the number of times a loop or other repetitive operation is executed.

CPU
See Central Processing Unit.

Data
Information that is processed and/or produced by the computer.

Decimal

A base ten number suitable for calculation.

Default

An assumption made by the computer or program when no specific value is given by the user. For example, if no literal value is assigned to an alpha field, it is automatically filled with spaces.

DIBOL

An acronym for Digital's Business Oriented Language. DIBOL is a computer programming language suited for the solution of business problems.

Direct Access

The process of directly obtaining or storing data from a location. Direct access makes a sequential search for data unnecessary.

End-of-file

The physical end of a file which is identified in various ways, depending on the computer system and programming language.

Execution

Performance of the instructions given by a program.

Field

A collection of individual characters or groups of characters in a record.

File

A collection of records which are stored, named, and treated as a unit.

Flowchart

A diagram showing the sequence of program operations, or the steps to be taken to solve a problem.

Hardware

The physical part of the computer.

Input

Data flowing into the computer.

Justify

The process of positioning data in a field which is larger in size than the data. In alpha fields, the data is left-justified and any remaining positions are filled with spaces. In decimal fields, the data is right-justified and any remaining positions are filled with zeros.

Literal

A alpha or decimal value that does not change; a type of constant.

Loop
A sequence of instructions that is executed repeatedly until specified conditions are met. A commonly used programming technique in processing data.

Object Program
A file which is output by the compiler.

Operation
The action specified by a single computer instruction.

Output
Data flowing out of the computer.

Peripheral Devices
Auxiliary pieces of equipment which help the CPU. They are used for permanent or long-term data storage, and as a means of communication.

Program
A sequence of instructions with which a computer can solve a problem.

Programming
The process of planning, writing, testing, correcting, and documenting the steps required for a computer to solve a problem or perform a series of operations.

Random Access
See Direct Access.

Record
A collection of related data fields in a file.

Relational Operator
A symbol used to compare two values. Some examples of relational operators are .EQ. (Equal), .LT. (Less than), and .GT. (Greater than).

Sequential Access
The process of obtaining or storing data relative to the location of the data which was most recently obtained or placed in storage.

Software
All of the written procedures and rules that control computer operations.

Source Program
A program written in a high level language such as DIBOL.

Statement
An instruction to the computer to perform some operation.

INDEX

INDEX (CONT.)

DIBOL for Beginners
Order No. AA-BI77A-TK
April 1984

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. Problems with software should be reported on a Software Performance Report (SPR) form. If you require a written reply and are eligible to receive one under SPR service, submit your comments on an SPR form.

Did you find errors in this manual? If so, specify by page.

_____

_____

_____

_____

_____

_____

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

_____

_____

_____

_____

_____

_____

Is there sufficient documentation on associated system programs required for use of the software described in this manual? If not, what material is missing and where should it be placed?

_____

_____

_____

_____

_____

_____

Please indicate the type of user/reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Non-programmer interested in computer concepts and capabilities

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code or Country _____

Please cut along this line.

—Do Not Tear - Fold Here and Tape—

digital

No Postage Necessary if Mailed in the United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
Applied Commercial Engineering MK1-2/H32
Continental Boulevard
Merrimack N.H. 03054

ATTN: Documentation Supervisor

Do Not Tear - Fold Here and Tape

Cut Along Dotted Line