DEC DATASYSTEMS 300 SERIES

INTRODUCTION TO DIBOL

Digital's Business-Oriented Language
A Programmed Text

digital

DEC-08-OCSTA-B-D

DIBOL

Self-Instruction Manual

(AN INTRODUCTION TO DIBOL)

For additional copies of this manual, order DEC-08-OCSTA-B-D from the Software Distribution Center, Digital Equipment Corporation, Maynard, Massachusetts 01754

First Printing, July 1972
Revised May 1973

CONTENTS

FOLDOUT ILLUSTRATIONS

INTRODUCTION

In this self-instruction text you will learn the fundamentals for using DIgital's Business Oriented Language - DIBOL. This compiler language is used to describe data-processing problems for the DEC DATASYSTEM 300 series computers. DIBOL is an integral part of the Series 300 Commercial Operating System - COS 300.

When you complete this course you should be able to program workable solutions to real problems in Billing, Accounts Receivable, Sales Analysis, Inventory Applications, and others. Given an application problem, you should be able to use DIBOL to develop a source computer program that can be compiled and run on the COS 300 System.

However, there are elements of the COS 300 software system that are not taught in this manual. You will find these in the COS 300 System Reference Manual (Order Number DEC-08-OCOSA-E-D).

This course presents sets of instructions, questions and answers in frames. Each question is followed by the correct answer. Fold-out pages are provided so that you may refer to examples of flowcharting, coding, problems, etc., as you progress. Additional problems are supplied to reinforce your knowledge through application. A summary is provided at the end of each chapter. A Glossary of terms is included for quick reference. Although this programmed instruction will give you a working knowledge of DIBOL, you should try to compile and run your first programs where computer facilities exist.

An interpretation of course completeness is based on our definition of a programmer - a person who prepares, or is responsible for, problem-solving procedures. The information presented in this text was prepared for these individuals, in a small to medium sized business environment, who of necessity may have to wear many "hats".

The programmer's hat cliche has significance for the student in that it suggests the proper way of approaching the subject information.

The programmer must be a specialist in more than one area. In his preparations he should assume the roles of a job and program analyst, a computer operator, and most important, the end user of the computer application.

The student should allow himself enough time to complete the course before attempting applications programming. In the final analysis, time and money will be saved by proper utilization of the computer.

DEVELOPMENTS IN SMALL BUSINESS DATA PROCESSING

Electronic Data Processing (EDP) technology has advanced to the point where most businessmen can realistically consider automating repetitious paper work. At present, thousands of small businesses are inundated with paperwork, payroll, inventory control, accounts receivable processing, and numerous other tasks that can be rapidly and accurately processed by computer.

Until recently, these businesses had no alternative to the service bureau since the cost of in-house computer configurations were prohibitively expensive. The total system power of a computer which would have cost a half-million dollars a decade ago can now be encompassed into a mini-computer desk configuration (DEC DATASYSTEM 300) for less than $30,000.

The technological milestones beginning in the 1960's were the shifts from vacuum tubes to transistors, then to Integrated Circuits (IC's), and most recently to Medium-Scale Integration (MSI). The size of a computer's Central Processor (CP) was directly related to these hardware advances.

The dramatic reduction in size and cost-per-circuit for the computer hardware is only part of the array of benefits. A new technology has emerged and matured as a direct result of millions of dollars worth of R & D resources expended by large corporations. This transition has been from tabulating equipment (unit record punched card tabulators - dedicated to wired single program operation) to the outstanding modularity and resultant flexibility of computer software.

Software is defined as computer control programs which are stored and/or loaded into the computer's memory banks. The program's instructions are automatically scanned by the computer's central processor to control job operation.

Not only has software emerged as an integral part of a computer configuration, but its usage has formulated specialized programs and procedures. These include some of the following:

> Implementation Language - programming language by which business data processing procedures may be precisely described in a standard form.
>
> System Monitor - a control program to supervise and verify the correct operation of all running programs including operator/system interaction.
>
> Utility Programs - standard service or housekeeping programs used to: sort and merge data, interchange programs and files which reside on various peripheral devices, update and build data files, edit and trace data, etc.

Combining the above, we have an operating system - an organized collection of techniques and procedures for operating a computer. For the DEC DATASYSTEM, they are part of a software package called the Commercial Operating System (COS). These job application software tools no longer require vast amounts of costly storage, nor does access to them run into minutes as it did with prior computer generations.

Using the hardware and standardized OS programs offered by Digital Equipment Corporation (DEC), the user can customize an application system by writing programs to conform to the way his company is conducting its business. He must carefully analyze his company's EDP requirements as a prerequisite to his system design. This methodical planning will contribute to a more rapid new system startup.

Initially his programs may be satisfactory, but they soon may become obsolete as procedures change to better utilize the computer. The in-house programmer should start slowly, developing the most convenient and practical programs first. He should also be prepared to continually modify programs and allow for new programs and system features. In this way he will be more responsive to the requirements of management.

The programmer should also make extensive use of the "conversational mode" type of program. Data should be entered online using the Cathode Ray Tube (CRT) display terminal (video screen with keyboard). The program should ask the operator for the required data, and then accept the answer. As much as possible, answer parameters should be built into the program. As an example, if the question from the computer is "number ordered?", an answer larger than the current stock level, or an answer that depletes stock below the reorder point, should produce a warning message in addition to the normal accounting within the program. Additionally, the program could produce an instant-echo for confirmation of back-order adjustments.

Since the interactive terminal allows the operator to check and make corrections to the data before it is transferred to the DECtape or disk, data can be visually validated. Obviously, conversational mode programs allow less-experienced operators to produce accurate records. A good system design will allow the existing clerical staff, under the guidance of the programmer (via displayed messages and operating procedures), to operate the computer. An accounts receivable clerk should continue doing receivables, while the person disbursing checks should continue doing that job. Employees can use the computer as a sophisticated tool to make each job easier and more pleasant.

The imaginative programmer can help job operation by preparing both batch and interactive programs and procedures. Batch processing is a sequential job stream procedure that uses an accumulation of related job units. It allows both data items and programs to be collected into groups for faster processing. Batch processing can be used to advantage for those cyclical or repetitious jobs. This technique will save the operator time while making maximum use of the computer.

COMMERCIAL DATA PROCESSING TERMINOLOGY

In order to describe and build better business systems, you should be familiar with the following applications terminology:

AUDIT - An operation or check designed to ascertain the validity of data. The validity of data is verified through the use of check sums, hash totals, maximums, minimums, redundancies, cross totals, and various other methods. The AUDIT is used to insure: that accounting records will not be destroyed, that the computer system will not incorrectly read or process data, or that someone will not manipulate data to produce wrong results. The report generated from an audit is called the AUDIT TRAIL, e.g., for a payroll application an AUDIT TRAIL for the validity of input data would contain error conditions, such as time records for terminated employees, employees with no time records, etc.

BACKUP - Pertains to equipment or procedures that are available for use in the event of systems failure and destroyed data files on tape or disk. The provisions for adequate BACKUP facilities and data files are an important factor in the design of all data processing systems, e.g., copy of a disk file on magnetic tape.

DATA BASE - Data records that must be stored in order to meet the information processing and retrieval needs of an organization. The term implies an integrated file of data used by many processing applications, in contrast to an individual data file for each separate application.

EAM - Electronic Accounting Machine, pertains to data processing equipment that is predominantly electromechanical such as: keypunch, collators, sorters. A computer is classified as EDP equipment. EAM equipment is also known as unit record or Tab equipment.

FIELD - A subdivision of a record containing one item of information, e.g., an employee's weekly time card containing his identification number in one FIELD.

FILE - A collection of related records. A FILE is usually either a transaction FILE or a master FILE.

FILE LABEL - A LABEL that identifies the FILE. An internal FILE LABEL is recorded as the first record of a file and is machine readable. A FILE LABEL is a control feature, e.g., insures that the operator has the proper master file for updating or prevents the operator from mistakenly using a master file as a scratch file.

FILE MAINTENANCE - The updating of a file to reflect the effects of non-periodic changes, such as adding, changing, or deleting data, e.g., addition of a new employee to the employee master file.

FILE PROCESSING - The periodic updating of a master file to reflect the effects of current data, usually transaction data contained on a transaction file, e.g., weekly payroll run that updates the payroll master file.

FIXED LENGTH
RECORD - A file containing a set of records, each of which contains the same number of characters. (Contrast with variable length records).

INDEX FILE - Pertains to a disk file that is organized somewhat like the books in a library, i.e., an index tells where the record is stored. The index contains two facts about each record in a file. First, the contents of the record's key field appears in the index. A key field contains data that uniquely identifies a record and is the basis for the file's sequence, e.g., customer number. The disk address represents the location on the disk where the record can be

found. An index label contains the same number of entries as there are records in the file.

Various terms associated with processing an indexed file are: index sequential processing, indexed access method, address routing method. These files may be processed sequentially or in random fashion.

INTEGRATED DATA
PROCESSING - Data processing by a stream that coordinates a number of previously unconnected processes in order to improve overall efficiency by reducing or eliminating redundant data entry or processing operations, e.g., a billing result file (data base) containing information from incoming customer orders is used for: inventory (calculated usage), accounts receivable (generate statements), and sales analysis applications. Integrated data processing is also known as management information systems (MIS).

KEY - One or more characters used to identify a particular record, especially used for sorting and merging operatings, e.g., an inventory part number and employee number. There may be multiple key fields in a record; e.g., a salesmen's commission file may be sorted by salesman within branch, within district, within region.

MASTER FILE - A reference file of semi-permanent information which is usually updated periodically by a transaction file, e.g., an employee MASTER FILE that contains a record for each employee. Each record would contain an employee number field, name field, address field, pay rate field, year-to-date gross pay file, etc. The year-to-date gross pay field would be updated each pay period.

RECORD - A group of related information items treated as a unit. A record is divided into one or more fields; e.g., an inventory record for each commodity might contain the following fields:

PRODUCT NUMBER (A FIELD)
DESCRIPTION
NUMBER ON HAND
NUMBER ON ORDER
MINIMUM BALANCE
UNIT COST
NUMBER USED YEAR-TO-DATE

RANDOM ACCESS
FILE - A mass storage device capable of accessing any record directly without processing all prior records. A data file arranged on a randomly generated record address -- access to a record is accomplished by calculation of a formula based on a key in record. No index is required for this type of file.

SEQUENTIAL
FILE - Pertains to a file where records are in ascending or descending sequential order by an identification key, e.g., inventory file sequenced by part number. SEQUENTIAL FILES are for batch processing in which the files are on cards or DECtape. However, disk files may also be sequential.

TRANSACTION
FILE - Records of data to be processed with master file record in order to update the master file, e.g., a file containing all of the daily transactions in an inventory control application, such as quality of items received, shipped or ordered, which update the inventory master file reflecting these changes. A transaction file is also known as a detail file.

COMMERCIAL DATA PROCESSING TERMINOLOGY (cont.)

VARIABLE LENGTH
RECORDS - A file containing a set of records in which the number of characters of each record may vary in length. Usually a VARIABLE LENGTH RECORD is preceded by the character count for that record.

As a small business computer with COS, the DDS300 provides solutions for users with varied business scopes and backgrounds. The implementation language for COS 300 is DIBOL. A general overview follows to provide the student and/or business manager insight into those system elements used for program creation.

DIBOL - A PROGRAMMING LANGUAGE

DIBOL - DIgital Business Oriented Language - is a general purpose higher level commercial programming language used by the programmer to implement commercial applications. Its compiler is an integral part of COS 300. With the COS 300 DIBOL compiler, the system generates application programs in computer machine language (MACRO instructions) to run on any DDS 300 computer.

A DIBOL program is divided into two sections, a data definition section and a procedure section. The data section states (tags) a data file's record information structure in program operable units. In the procedure section, the language consists of a select group of English-like procedural verbs, each with comprehensive arguments. The verbs: PROC, START, END, ON ERROR, INIT, FINI, INCR, TRACE, NO TRACE, TRAP, XMIT, READ, WRITE, GO TO, IF, CALL, RETURN, CHAIN, ACCEPT, DISPLAY, FORMS, and STOP, plus data manipulation statements, provide the user with easy to use and powerful statements for the development of his programs.

There are eight types of statements:

Compiler Statements which tell the compiler the nature of the statements to follow.

Device Control Statements which prepare data files (open and close) for use by the application program.

Data Specification Statements which describe the type, size and location of data elements.

Data Manipulation Statements which control calculations and movement of data within memory.

Data Accept and Display Statements which define the cursor coordinates used to format data entry and display on the CRT terminal.

Control Statements which govern the sequence of execution of statements within a program.

Input/Output Statements which control data movement within memory or between memory and peripheral devices.

Debugging Statements which trace program execution during test runs.

Language Features

Simple English-like Procedural Statements. Meaningful expressions to the user and the system's program compiler (not assembly language).

ANSCII Character Sub-set (specified by the American National Standards Institute. ANSCII character code used as a Standard Code for Information Interchange).

Multi-I/O-Level Data Access by File, and Record. Direct access, at the logical (program) level, to data stored on disk or DECtape.

Data Manipulation via: Record, Field and Subfield. Statements to clear data fields, move data between fields, convert decimal data to/from alphanumeric data, and format data, etc.

Arithmetic Expression. Performs division, multiplication, addition, subtraction, and rounding.

Array Handling. Any part of an array (series of items) can be accessed in a program statement by listing the position of the item. Subscripting notations (expressions in parentheses) are used to specify items in a list or table according to DIBOL rules.

File Initialization. Statements assigning specified peripheral Input/Output channels to logical or physical devices.

Branching. A computer operation similar to switching, where a selection is made between two or more possible courses of action depending on a related fact condition (IF and GO TO statements).

Many Levels of CALL Nesting. Statements which include routines to call other routines.

Tracing. Trace statements may be placed at strategic locations within a program to provide a usage correlation (source line numbers) between statement execution and the intended source program logic.

Editing. When transferring data, field editing occurs with left and right justification, padding and "check protect" features.

Cursor Control. Statements which provide display and data entry in a particular applications format on the CRT Terminal.

Forms Control. The Forms statement is used to automatically position business forms to be printed on the DEC DATASYSTEM Line Printer.

COMMERCIAL OPERATING SYSTEM

COMPILER

The COS 300 Compiler enables the DIBOL user to compile a source program utilizing up to 28K word locations of memory (56,000 characters) for his application system. Input for the source program can originate from the console keyboard, from cards, from DECtape or disk, and from paper tape. Source program inpur is implemented via the COS 300 Monitor which provides the user both input editing and generation of job control statements. The standard output from compilation resides on the System's mass storage device in executable format and may be stored by name in a user's program library.

As a mass storage resident system, DIBOL provides the facilities for random storage and direct retrieval of programs and data on both DECtape and cartridge disk. The system also provides the ability to dynamically divide DECtape and disk storage into fifteen logical units for data file storage.

Each cartridge disk can contain up to 404 directly accessible segments of 8,000 bytes each. The COS 300 system handles storage capacities ranging from 377,344 character DECtapes to 3.2 million character disk cartridges. This allows a simple but comprehensive means for new users to utilize state-of-the-art cartridge disk storage for their on-line data base.

At compile time, the minimum configuration required to operate is the DEC DATASYSTEM Model 320 and resident COS 300 software. At run time the user's applications programs can utilize a wide range of input/output facilities, full internal capacity, and through-put of any model in the DDS 300 series. This includes Models 320, 330, and 340.

Several COS 300 system programs are utilized with the compiler in the process of creating user programs.

Monitor

COS 300 provides program operation master control via a System Monitor. To facilitate memory economy the Monitor resides in two segments: one core resident, and the other residing on the system device. Together these segments provide the following facilities through a comprehensive set of Monitor commands used for:

- Program Loading,
- Editing,
- File Directories,
- Operation Messages.

The Monitor contains all the system I/O handlers required for efficient throughput and a high degree of program/device independence. The system provides a specialized software handler for each peripheral device on the DEC DATASYSTEM.

The assignment of logical units to physical mass storage devices provides greater utilization of the storage area. This device independence is available at run time. Any mass storage device can be specified for I/O and program execution using the devices specified via SYSGEN.

Editor

Editing consists of a line editor as part of the Monitor. It is an operator/system interactive editor providing a "scratch-pad file" for source program entry. Input statements consist of line numbers followed by the information to be inserted, deleted or changed. The COS 300 editor provides automatic sequencing and resequencing of line numbers by simple commands. Input for the editor can originate from the console keyboard, cards, paper tape, DECtape, or disk. Output from the editor can be a listing of a file on the console display or the line printer or paper tape. In the program development stage the user can save and quickly recall programs from the system device (DECtape or cartridge disk). In an operational mode, the user can batch commands to the Monitor into a file to be executed as a job stream.

COMMERCIAL OPERATING SYSTEM (cont.)

SYSGEN (SYStem GENeration)

A conversational utility program that allows the user to configure or modify the current system using simple English-type statements. It provides the following optional features:

Configures the I/O handlers in the system,
Takes new logical unit assignments from the operator's terminal,
Prints a table of current logical unit assignments.

The user can specify the type of line printer used and where the system is to reside, on DECtape or disk. The user can also specify the number of columns used on the line printer, either 80 or 132 columns. SYSGEN provides the facility to transfer the system to another device for installation startup.

PIP Peripheral Interchange Program

A utility program which provides file transfer from one device to another. It will permit the user to move source, binary, system, or data files from one device to another. It has the following capabilities:

- Replaces the old file with a new file
- Transfers Input from cards, paper tape, disk, or DECtape, and Outputs to paper tape, DECtape, disk or the line printer
- Copies an entire DECtape or disk onto a similar device
- Eliminates overhead space from the file directory.

Data File Creation and Maintenance Programs

System programs available for structuring transaction files are: BUILD, UPDATE, and SORT. For more detailed information concerning this software, refer to the COS 300 System Reference Manual.

BUILD

A file creation program used to create a data file. It is a key-word data entry package. A BUILD "Control Program" allows the user to specify key words followed by an ordered string of formatted data. BUILD has the following features:

- Provides hash totaling,
- Provides range checking,
- Computes check digits,
- Provides auto-dup field (automatically duplicates fields),
- Permits specification of default fields,
- Permits specification of incremental fields,
- Checks errors on any one or all given fields within a key word line,
- Sets special field flags that the user can later check under program control,
- Has the ability to specify up to seven different output files from one input file.

BUILD facilitates a way of flagging certain fields within a record for use as a program control switch.

UPDATE

A master-file maintenance program used to:

- Change existing records on the data file,
- Insert new records,
- Delete old records,
- Print a report showing all changes, inserts and deletions.

SORT

COS 300 SORT is a poly-phase sort. It can sort data file records in ascending and descending order. SORT requires a minimum of three DECtape units or an equivalent disk unit. The user can specify up to eight fields (with sub-fields) of a fixed length record as a sort key.

The SORT has a merge file capability. For a multi-reel sort, each file must be sorted then merged. The same SORT control program may be used for both sorting and merging.

Utility Programs

Several utility programs are provided, among which are the following: CREF, DAFT, MARK, BOOT, and PATCH.

CREF

CREF is a cross reference utility program to aid program development. It provides an alphabetical listing of all symbols used in a DIBOL program, along with the line number where each symbol is defined and used.

DAFT

The DAFT (Dump And Fix Technique) program provides the capability to search for, examine and change records as well as to list records and parts of records on the line printer or terminal.

MARK

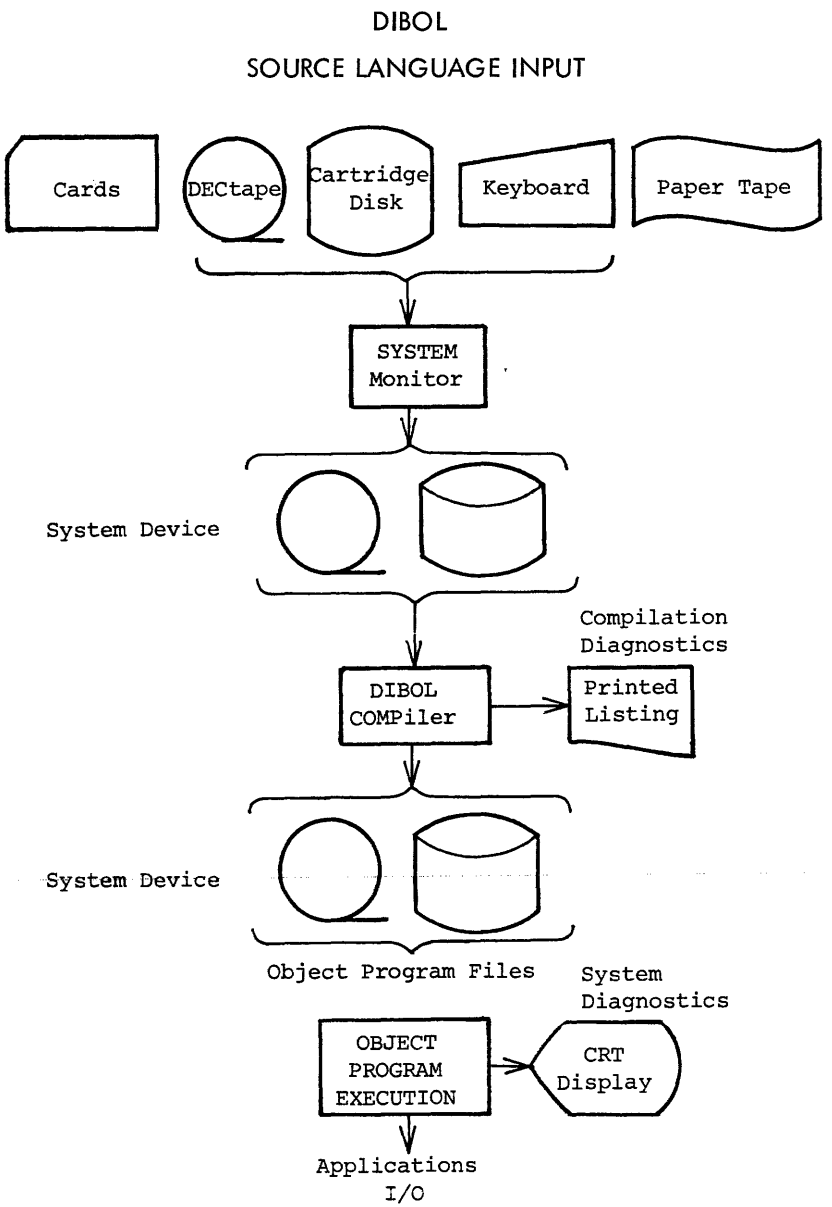
There are four format programs, RKEMRK, RK8MRK, TDMARK, and DTMARK, which are used to mark DECtape and disk for use with the COS 300.

BOOT

BOOT is used to bootstrap the system from one device to another. For example, BOOT is run to transfer control from DECtape to disk so that the latter may become the system device.

PATCH

PATCH is used to fix (or patch) system programs or the Monitor on a COS system tape. Data to make the changes is provided by DEC as patches in the form of a dialog.



MANUAL AND SYSTEM CONVENTIONS

Several documentation symbols and terms used in this text are described below-

Symbols

Lower-case characters - Represent information that must be supplied by the user, such as values, names and other parameters.

Upper-case characters - Words or characters that must be used exactly as shown.

Ellipsis... - Indicates the optional repeating of the preceding data.

Underscored characters - Indicate output from the system.

␣ (Space) - Indicates a space.

{} (Braces) - Braces indicate a choice of one of the items enclosed.

[ ] (Brackets) - Brackets indicate an optional feature.

↵ (CR Key) - The down-arrow indicates a Carriage Return Key operation on the terminal keyboard. At this point, control is advanced to the next line.

Terms

file-name, program-name, label and keyword - are used to identify names assigned to files, programs, statements, and input lines. These names may be of any length, but only the first 6 characters are recognized.

cursor - An underscore symbol on the operator's display screen which indicates the character position for the next keyboard stroke.

SECTION 1

Basic Source Language Programming

This chapter is intended to give the student a frame of reference in the form of an overview of DIBOL programming and operation.

In the following frames you will learn, through programmed instruction, how to read a simple DIBOL program. The questions will be based on program examples which provide frame answers.

Frame one presents a program statement of what the first example program does. (Turn to Foldout #1). Read the program definition and review the foldout before proceeding to the QUESTION and ANSWER.

PROGRAM: Information is stored in records located on a DECtape. Each record is to contain 64 characters. A read operation reads one record at a time (starting with the first record in the file). List on the line printer all information on the DECtape file, printing one record on each line. After printing the last record on the tape file, stop the program.

Systems and program flow charting is a technique used in organizing and documenting information about existing application systems and in planning new ones. Diagrams called flowcharts show the flow of information and the sequence of operation. They are important items in evaluating procedural logic and useful tools for future program expansion and revision.

---

1. QUESTION: Does the flowchart on Foldout #1 illustrate the logic outlined in the verbal statement of our program?

***

ANSWER: Yes, the flowchart is accurate.
The shapes of the boxes in the flowchart denote different functions such as comparison, reading/writing, beginning/end, and internal data arrangement. Using the flowchart, answer the questions:

---

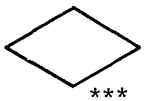
2. QUESTION: What is the function of this symbol?

***

ANSWER: It denotes the beginning/end of the program logic flow.

---

3. QUESTION: What is the function of this symbol?



***

ANSWER: It denotes the testing of a condition and, depending upon the outcome of the test, shows the action to be taken.

---

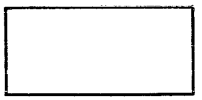
4. QUESTION: What is the function of this symbol?

***

ANSWER: It denotes an Input (reading) or an Output (writing) operation to be performed by the computer. (In the program there is an internal device assignment so the computer would issue a read/write command to the proper input/output device.)

---

5. QUESTION: What is the function denoted by this symbol?



***

ANSWER: It denotes explicit commands such as device initialization, move data, etc.

Below the flowchart on Foldout #1 is the DIBOL-coded program which accomplishes the functions diagrammed. Note that it requires only ten statements to accomplish the outlined task.

The following dialogue is designed to help the reader understand the function of each statement in the DIBOL program.

---

6. QUESTION: From the flowchart, is each character of information passed directly from the DECtape to the printer?

***

ANSWER: No. A complete record composed of 64 characters is read into the computer memory before any data is written (output) on the line printer.

---

7. QUESTION: Since data records are not restricted in length to 64 characters, how does the computer know how much memory to reserve for the storage of the data record?

***

ANSWER: The programmer must tell the computer how much memory will be required to store input data. A DIBOL RECORD (or BLOCK) statement reserves areas of core to be used during processing and as temporary storage (input/output buffers).

---

8. QUESTION: In a program the area of memory reserved for record storage precedes the processing instructions. From the DIBOL coding on Foldout #1, which statement allocates 64 characters of storage?

***

ANSWER: Statement 3; (FIELD1, A64).

---

9. QUESTION: Statement number 3 says the block of storage labelled FIELD1 will be reserved for 64 alphanumeric (A) characters. What is FIELD1?

***

ANSWER: It is a field label. It could just as well be called XX, YY, or any six character field beginning with a letter. It serves as a symbolic name which the programmer can reference from the procedure section of the program (note: it is referenced in the sample program via the record's label). One or more field statements is required for each RECORD statement.

---

10. QUESTION: The RECORD statement (#2) gives a label "NAME" to an area of computer memory available for record input. If several different input devices are being used, several different RECORD statements (with their respective field-definition statements following) could appear. In order to designate two 16 alphanumeric character fields and one 32 alphanumeric character field instead of the present 64-character field, write the appropriate RECORD and field-definition statements.

***

ANSWER:

```
RECORD NAME2
    FIELD1, A16
    FIELD2, A16
    FIELD3, A32
```

---

11. QUESTION: A compiler control statement is a non-executable DIBOL statement. Such a statement gives the compiling program information necessary to properly interpret notations made by the programmer. Compiler statements tell the compiler program when to begin and end encoding DIBOL source statements, and when to begin converting DIBOL statements into actual machine procedures. Look at the sample DIBOL program and determine which are the compiler statements.

***

ANSWER: Statements 1, 4, and 10. Statements 1 and 10 tell the program the bounds of the DIBOL coding. Statement 4 tells the compiler program to interpret the following lines of coding as procedure to be executed by the computer.

---

12. QUESTION: The data section of a DIBOL program describes the data elements used in the program and allocates memory. Which statements in Foldout #1 comprise the data section?

***

ANSWER: Statements 2 and 3 comprise the data section.

---

13. QUESTION: Which statements are actual processing instructions?

***

ANSWER: Statements 5 through 9 are processing statements.

---

14. QUESTION: What is the function of the word LOOP in statement 8?

***

ANSWER: LOOP is a label denoting a point in the processing cycle to which the program branches. In this case, the program branches to statement 6.

---

15. QUESTION: Statement 5 is the first processing instruction. From the flowchart, what is accomplished by this statement?

***

ANSWER: It tells the computer which channel number will be used (whenever the program refers to this file, its channel number will be 2); whether the file will be read (Input) or written (Output), the name of the file (FILE1), and where the file could be found (logical unit 1). This process is called file-initialization.

Depending on the logical unit assignments made through program SYSGEN, FILE1 could be on DECtape or disk.

---

16. QUESTION: From the flowchart (and following the INIT statement in the program), what does the XMIT statement (#6) do?

***

ANSWER: It causes a read operation from channel 2. The XMIT refers to data transmission. It can either read (IN) or write (OUT) depending upon the file-initialization. If a file is initialized as an input (IN), the XMIT statement would cause a read operation from the file. If a file is initialized as an output (OUT), the XMIT statement would cause a write operation onto the file. The data is either read from, or written into, the RECORD specified in the XMIT statement (in this example, RECORD NAME).

---

17. QUESTION: For XMIT (2, NAME) to cause a record to be written, what would the initialization statement look like?

***

ANSWER: INIT (2, Output, file-name, unit)

---

18. QUESTION: For XMIT (2, NAME, ENDFIL) to cause a record to be read, what would the initialization statement look like?

***

ANSWER: INIT (2, Input, file-name, unit). (As in statement 5 in our example.)

---

19. QUESTION: Statement 6 says: Read a record from channel number 2, storing the data from that record in the area labeled NAME. When no more records are available, i.e., the end-of-file (EOF) has been reached, then jump to the instruction labeled ENDFIL. Noting that only READ instructions have pointers to which the program will branch when an EOF is reached, what does statement 7 do?

***

ANSWER: Since there is no end-of-file pointer (just a channel number and the RECORD label) this must be a write command (writing data from storage RECORD NAME to channel number 6).

---

20. QUESTION: Note that it is necessary to initialize only a file-oriented device such as DECtape or disk. Devices such as terminal, line printers, cards, and paper tape readers do not need initialization. Is device specified by channel number 6 a file-oriented device?

***

ANSWER: No. It is a line printer and therefore does not need initialization.

If not specified, the following channel number associations are assumed by default:

5	paper tape reader
6	line printer
7	keyboard
8	terminal scope or printer

---

21. QUESTION: What does statement 8 do?

***

ANSWER: It is an unconditional command for the computer to branch to the instruction labeled LOOP.

---

22. QUESTION: What is ENDFIL?

***

ANSWER: It is the label for the end-of-file routine referenced in statement 6. This label indicates the location for program transfer at the end of the input file. In this example, program control would transfer to statement 9.

---

23. QUESTION: What seems to be the function of statement 9?

***

ANSWER: It is a FINIsh statement with respect to the file on channel 2. Actually, in an output file, an end-of-file mark would be put on the tape, the tape would be rewound, and the channel number freed for other use. In the case of our input file, the channel number is freed but the file is not rewound. Only file-oriented devices require a FINI statement.

---

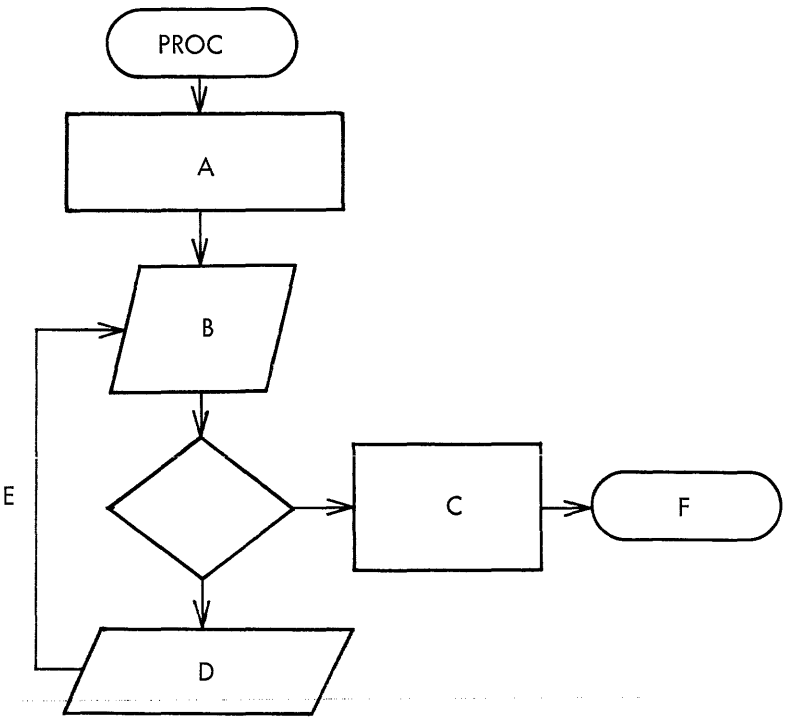
24. QUESTION: Why was channel 6 not issued a FINI command?

***

ANSWER: It is not a file-oriented device. The only file-oriented devices on a DIBOL configuration are DECtape and disk.

---

25. QUESTION: Below is the same flowchart diagram as listed on the foldout. Mark in the appropriate statement number corresponding to the flowchart function.



***

ANSWER:
A=5
B=6
C=9
D=7
E=8
F=10

digital EQUIPMENT CORPORATION
MAYNARD, MASSACHUSETTS

COS 300 DIBOL CODING FORM

PROGRAM TITLE	REQUESTED BY	PAGE OF
PROGRAMMER	DATE	PROJECT ID

1 7 9 17 25 33 41 49 57 65 72

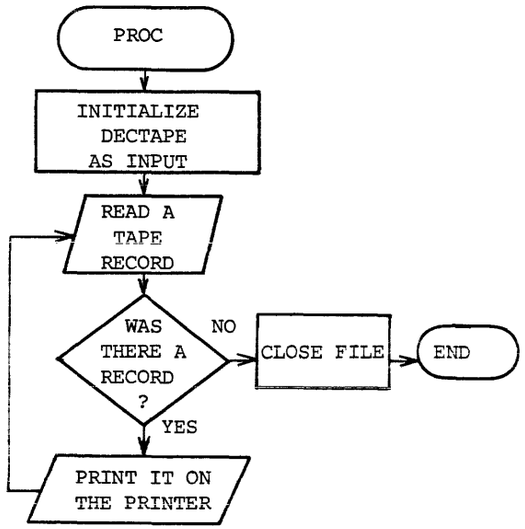
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72

DEC 7 - (302) - 1134 - N172

FOLDOUT #1

SAMPLE PROBLEM

FLOWCHART



DIBOL PROGRAM FOR PROBLEM

```
                                        Statement Numbers
START                                   ;1  COMMENTS
RECORD NAME                             ;2
       FIELD1, A64                      ;3
PROC                                    ;4
       INIT (2,IN,'FILE',1)             ;5
LOOP,  XMIT (2, NAME, ENDFIL)           ;6
       XMIT (6, NAME)                   ;7
       GO TO LOOP                       ;8
ENDFIL,    FINI (2)                     ;9
END                                     ;10
```

SUMMARY

Sample Problem

To review the initial problem, i.e., printing a 64 character record from a tape file onto a line printer until the end-of-file, examine the following lines of the DIBOL program:

```
1     START                    ;Compiler statement, non-executable.
2     RECORD NAME              ;Indicates the beginning of the con-
                               tiguous area for the data elements
                               that comprise RECORD NAME.
3         FIELD1,  A64         ;Data statement FIELD1 is an alpha-
                               numeric field 64 characters long.
4     PROC                     ;Compiler statement - begins pro-
                               cedure section.
5         INIT (2,IN,
          'FILE1',1)           ;Initialize channel 2 as an input file.
                               The label of the file is FILE1 and it
                               can be found on logical unit 1.
6     LOOP,XMIT
      (2, NAME,ENDFIL)         ;Read a record from channel 2 into the
                               area assigned to RECORD NAME.
                               When end-of-file is reached, it
                               causes a program transfer to ENDFIL.
7         XMIT (6,NAME)        ;Write RECORD NAME onto channel
                               6 (line printer).
8         GO TO LOOP           ;Go to statement that reads another
                               record.
9     ENDFIL,FINI (2)          ;end-of-file.
10    END                      ;Compiler statement--indicates the
                               end of program.
```

BASIC OPERATING STEPS

There are several fundamental operating steps required to convert your documented program logic into machine usable binary code. Using a properly activated (powered-up) DEC DATASYSTEM Computer, you must first load the central processor's memory with the COS 300 Monitor program. This system initialization is activated by the Hardware Bootstrap Switch, but only after you have correctly installed the System DECtape or Disk Cartridge on the drive addressed as zero. Refer to steps 1 and 2 in Figure 1-1. This figure presents the basic operating steps as: major operating categories, an operational flow diagram, and specific operating steps (1-10). The major operating categories are as follows:

System initialization,
Keyboard input,
Source program editing,
Source program compilation,
Source program syntax evaluation and correction,
Program logic testing and correction,
Object program storage,
Source program storage.

Study Figure 1-1 and then answer the following questions.

The Monitor routine must be loaded via a bootstrap operation. This is an internal computer hardware/software technique designed to bring itself into a desired state of readiness by means of its own action, e.g., a hardware initiated routine whose first few instructions are sufficient to bring the rest of itself into the computer from either the DECtape or disk storage device.

---

26. QUESTION: From Figure 1-1, what operator step or steps are necessary to accomplish this initialization?

***

ANSWER: Steps 1 and 2 load the System Monitor.

NOTE

In addition the operator must enter the current date before proceeding. This date is used during program execution to date reports, files, and new programs.

Review step 5 in Figure 1-1. The Monitor provides editing commands to input and manipulate Source Program statements in a temporary storage area (edit buffer) within memory. They include:

Number commands - inserts the text line beginning with the number into the edit buffer (line number text).

Line Number (LN) - automatically outputs line numbers so new programs can be entered without typing each line number.

Erase (ER) - erases text from the edit buffer.

Resequence (RE) - renumbers the program lines to adjust for additions and deletions.

All text input to the Monitor must be assigned a series of line numbers All inserts, changes, and deletions are accomplished using these numbers.

---

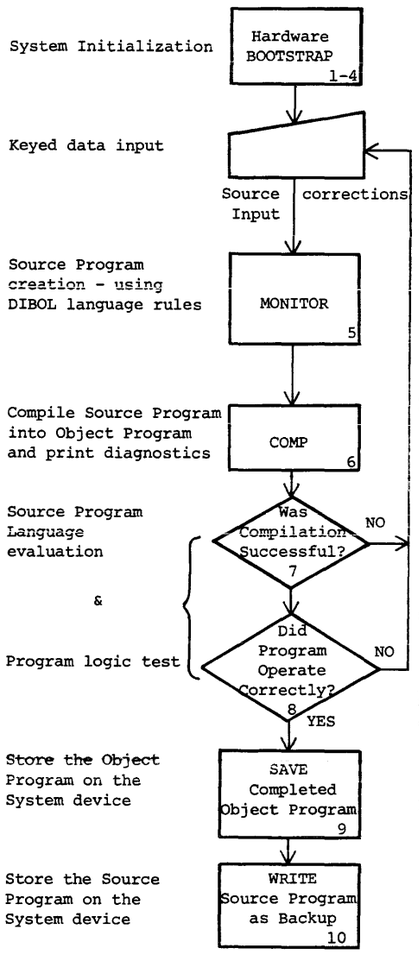
27. QUESTION: Using Figure 1-1, which of the above commands is used as a prerequisite to program text entry?

***

ANSWER: The ERASE command, when used without line numbers, clears the entire edit buffer. This prevents unused buffer lines, containing lines from a previous program, from appearing as part of the current program.

FOLDOUT #2

Basic Operating Steps



System Initialization

1. Mount the proper System DECtape or Disk Cartridge.
2. Press the bootstrap switch.
3. The Computer responds with DATE?
4. The operator enters the month/day/year in the following statement. .DATE␣mm/dd/yy) After the system responds on a new line with an initial period (.) enter ERASE) to clear the Edit Buffer.

Editor Source Input

5. After the system responds on a new line with an initial period (.), enter a DIBOL program putting one statement on a line. Precede each statement with a line number and a space. To correct a statement, type its line number, a space, and the new statement.

Compile Source Program

6. To compile the completed program in the editor scratch area enter: .RUN COMP)
7. Check the source listing generated by the compiler for error messages. A caret (^) points to the error. Use the Editor functions to correct all language errors and then recompile.
8. To run the compiler program if no errors are detected enter: .RUN)

Program File Creation

9. To save the object program under a unique program name enter: .SAVE progrnm)
10. To save the source program from the editor scratch area for future modification enter: .WRITE progrnm)

Figure 1-1

To compile the completed program use the Monitor command statement RU␣COMP↵.

The compiler takes the DIBOL language program from the edit buffer and converts it to an object program which can be executed by the computer. In the process of creating the executable program the compiler can generate a printed listing of the source program and a storage map of the records and fields which are used by the program.

The compiler checks the source program for DIBOL syntax errors. The source program must be free of these errors before object program code can be generated.

Most compiler error messages are printed on the source listing after the line in which the error occurs. A caret (^) in the source line points to the approximate location of the error. A complete list of errors can be found in the COS 300 System Reference Manual. Several commonly found messages follow:

UNDEFINED NAME
NAME PREVIOUSLY DEFINED
BAD ALPHA VALUE
BAD DECIMAL VALUE
COMMA MISSING

---

28. QUESTION: From Figure 1-1, when an error message occurs at step 7, what steps must be taken to correct it?

***

ANSWER: Steps 5, 6 and 7 in Figure 1-1 must be repeated to correct the error. You should correct all known errors before step 6.

The COMPiler will signal a clean listing with the message "NO ERRORS DETECTED" on the print-out. This indicates that your usage of the DIBOL language syntax is correct. It does not guarantee that the logic of your program will produce the desired results. The only way to test your program logic properly is to run the compiled program with real or simulated data.

To run the compiled program, enter the Monitor command RUN↵. When a file-name is not given after RUN, the binary file (compiled source program) in the Edit Buffer is used as input to be executed.

---

29. QUESTION: From Figure 1-1, when the program fails to perform properly with test parameters what steps must be taken to modify it?

***

ANSWER: First make the necessary adjustment to the program documentation then steps 5, 6, 7 and 8 must be repeated to change the program.

The Monitor allows both the binary file (B) and source file (S) to be assigned the same name. To protect both versions of the completed program, you just assign it a unique program name and transfer the edit buffer to a storage device before a new program is entered. This is accomplished by two Monitor commands, SAVE and WRITE.

---

30. QUESTION: From Figure 1-1 which of the above mentioned Monitor commands is used to save the compiled binary program?

***

ANSWER: The SAVE command stores the finished version of the object program on the System storage device under a unique program name.

SECTION 2

DIBOL Syntax

The student should now have a general knowledge of the elements that make up a DIBOL-coded program, thus making the information in this section more meaningful.

START
PROC
END

DATA SECTION

There are two sections in a DIBOL program. First is the data section which describes all data and causes allocation of memory storage. Second, there is the processing section which contains the executable instructions.

---

1. QUESTION: What is the statement that separates the data section from the processing section?

***

ANSWER: The PROC statement.

---

2. QUESTION: Is PROC an executable statement? If not, what is it?

***

ANSWER: PROC is not an executable statement. From the previous section we recognize it as a compiler statement.

---

3. QUESTION: What is a compiler statement?

***

ANSWER: A compiler statement is a message to the compiler program indicating the nature of the DIBOL-language statements. A compiler instruction is not executable by the DIBOL program.

There are three kinds of statements in a DIBOL program:

1. Compiler statements.
2. Data statements.
3. Procedure statements.

DIBOL programs normally consist of a START statement, followed by the data section (composed of data statements), followed by a required PROC statement (a compiler statement), followed by the procedure section (composed of procedure statements), followed by an END statement. START and END are optional statements.

---

4. QUESTION: What is the only required compiler instruction in a DIBOL program?

***

ANSWER: PROC

---

5. QUESTION: What are the three compiler statements and two sections that make up a DIBOL source-program (in the order in which they appear)?

***

ANSWER:
START (compiler instruction)
data section
PROC (compiler instruction)
processing section
END (compiler instruction)

RECORD

6. QUESTION: Where is the data section in a DIBOL program?

***

ANSWER: The data section is between the START and PROC statements.

---

7. QUESTION: From the sample program, in which section does the RECORD statement reside?

***

ANSWER: The RECORD statement resides in the data section.

The RECORD statement designates the beginning of a group of data statements. It may or may not give that group a name. It controls where in memory the block of data will be stored. A RECORD statement must be followed by one or more data statements. (A data statement defines all data elements with respect to type and size.) The word BLOCK may be used interchangeably with RECORD. The general format for a RECORD statement is:

RECORD	record-name,	X
required	optional	optional

---

8. QUESTION: In the following example, is the RECORD statement used correctly?

```
START
RECORD A
RECORD B
      B1, A6
PROC
```

***

ANSWER: RECORD A is an invalid statement because a RECORD statement must be followed by one or more data statements. RECORD B is used correctly.

A block of data requires a name only when referenced by an XMIT (data transfer) statement. A record may be read and stored in this area, or the contents of this area may be written (output). There is no punctuation between a RECORD statement and its name (if a name is required).

---

9. QUESTION: What can be deduced about the second RECORD statement following?

```
START
RECORD A
        A1, A6
        A2, D1
RECORD
        B1, D3
PROC
```

***

ANSWER: Since the second RECORD statement does not have a name, it is not intended to be used as an input/output buffer. It is used only for temporary storage of program data.

The data statement is used to define all data elements with respect to type and size. The DIBOL compiler assigns storage for the data on the basis of these statements. Any data statement that follows a record name is assigned to the contiguous memory locations in the order that the element occurs. If a record name is missing, the succeeding data statements are assigned to contiguous locations but not associated with any record name for input/output. If such data statements are referenced, they are done so individually.

The general format of the data statement is:

data name,	data specification,	initialization-specif.
optional	required	optional

The data name is optional, that is, a comma may be used without a data name if the program does not reference that individual data element but only references the entire RECORD. This is convenient when formatting an output line for the printer, so that intercolumn spaces do not require a data name but merely a comma followed by the type and size, e.g., (,A5). Normally, the data name is used, followed by the data specification (type and size), with an optional initialization field.

The initialization-specification would normally be used in the temporary storage record but could also be used in an output record. If a specific data element is to be referenced, it must have a data name.

Following are examples of valid data statements:

```
A, A10
A, A7,'DIGITAL'
A, D6, 123456
FISH, A4, 'FISH'
    , A5
COST, 5D6
```

NOTE

The Data element COST consists of 5D6, which means there is an array of five fields, each six digits long. This could have been written as:

```
COST1, D6
COST2, D6
COST3, D6
COST4, D6
COST5, D6
```

(The referencing of COST, 5D6, is done with subscripts which will be defined in the procedure section.)

---

10. QUESTION: In the following statement, what are the fields?

TOT, D6, 000012

***

ANSWER: TOT is the name by means of which the data can be referred; D6 is the data specification (in this case, six decimal digits) and 000012 is the initialization specification (setting the six decimal digits to an initial value of 12).

The data specification field (which follows the data name) consists of a data type (Alpha or Decimal), and the data size in characters. If the data size is omitted, 1 is assumed. If the initialization specification is present, the data specification is followed by a comma, then an alphanumeric or decimal constant. The alphanumeric may contain any legal character enclosed in apostrophes. The decimal constant is a string of digits, optionally preceded by a plus or a minus sign. The plus sign is implied and the minus sign does not require a character for storage; i.e., NUM, D5, -12345. Data types cannot be mixed. For example, an alphanumeric constant may not be assigned to a decimal variable. The data element is assigned to the value of the initialization specification at the beginning of program execution. If the initialization specification is omitted, an alpha field is set to spaces and a decimal field to zeroes.

---

11. QUESTION: Describe the data specified for the following five items:

a) A, A10
b) TOT, D6
c) NUMS, 10D3
d) HDRS, 10A12
e) TABLE, 3D2, 13, 15, 18

***

ANSWER:

a) A is an alphanumeric element with ten characters having a value of spaces.
b) TOT is a decimal element with six digits having a value of zero.
c) NUMS is an array of ten decimal numbers, each with three digits having a value of zero.
d) HDRS is an array of ten alphanumeric elements, each with 12 characters having a value of spaces.
e) TABLE is an array of 3 decimal numbers, each with two digits. The first element has a value of 13, the second a value of 15, and the third a value of 18.

---

12. QUESTION: Suppose you want to reserve, in computer memory, a place to store a record with four fields to be described as follows: FLD1 has three alpha characters, FLD2 has six decimal digits, FLD3 has four decimal digits set to the value 125, and FLD4 has 10 alpha characters set equal to the name DIGITAL. Write the appropriate RECORD and data specifications for this input buffer.

***

ANSWER: RECORD IN (any name will do)

```
FLD1,A3
FLD2,D6
FLD3,D4,0125
FLD4,A10,'DIGITAL ␣␣␣ '
```

There are three ways data items can be initialized in the Data Section.

1) A Data statement containing a P.

2) A Data statement containing a D.

3) A Data statement containing an initialization specification value (previously discussed).

One way of initializing a data item is by putting a P immediately after the data specification field of a data statement. Upon loading the program, the computer will ask the operator (via the terminal) to enter the value he wants to give that data item. A common use of this feature is to obtain a report date which differs from today's date in Monitor. For example, the data statement might be described as:

```
DATE,A8,P
```

Upon loading the program, the following message would type out on the console:

```
ENTER DATE
```

At this point the program would wait for the operator to type in eight characters and type carriage return. The operator might type in 07/07/72 and type carriage return. CAUTION: If less than eight characters were entered, the results would be stored left-justified in DATE. If DATE were a decimal field, the results would also be stored left-justified. Unentered characters are either zero or spaces, depending upon the field type. Too many characters would run over into the following fields. For decimal fields, there is no verification that the characters entered were decimal characters.

---

13. QUESTION: Suppose the programmer wanted a three-digit customer-number to be supplied by the operator at the beginning of the run. Write a data statement to initialize a field named CUST.

***

ANSWER: CUST,D3,P

Ordinarily, input from the keyboard must be described in alpha format. This restriction does not hold true for initialization data (P).

D OPTION

A second way of initializing a data item is by putting a D immediately after the data specification field of a data statement. When the program is run, today's date (as specified to the Monitor at start up) is automatically stored in any field having a D in its data field specification. For example, the statement might be described as:

TODAY, D6, D

The date in Monitor would automatically be stored in the form MODAYR.

---

14. QUESTION: a) Write a data statement to initialize a field named RNDATE to the date stored in Monitor.

b) What would RNDATE contain at run time if an operator previously entered July 4, 1972 as the Monitor date?

***

ANSWER: a) RNDATE, D6, D
b) 070472

A third way to set the value of a data item is by using the initialization option, i.e., specifying an alphanumeric or decimal constant. The alphanumeric may contain any legal characters enclosed in apostrophes. The decimal constant is a string of digits, optionally preceded or followed by a plus or a minus sign, but without apostrophes. Data types cannot be mixed in that an alphanumeric constant (i.e., DEC, D3). But '004' can be assigned to an A3 alphanumeric field.

---

15. QUESTION: Are any of the following statements incorrect?

a) A,A8,'ABCDEF98'
b) TOT,D3,'123'
c) NUM,A3,123
d) B,D6,222334
e) C,D3,23A
f) D,D3,456-

***

ANSWER: Statement b is incorrect because an alphanumeric notation (an apostrophe) was used to enclose a decimal item.

(c) is incorrect because a decimal notation was used to describe an alphanumeric constant.

(e) is incorrect because 23A is not a valid decimal number.

Statements a, d, and f are correct.

When an initialized value is specified in a data statement, its length must correspond to the length of its respective data statement, for example, NUM, D4,0070. It would be illegal to initialize NUM to

70 since NUM was defined as a D4 field. The initialization specification does not insert leading or trailing blanks (zeros). DIBOL will not permit the size of the initialized value to differ from the data field size.

---

16. QUESTION: Which data statements are incorrect?

```
a) A1,A8,'ABCDEF641'
b) COST,D4,7779
c) QTY,D5,'10000'
d) NUM,D7,59796
e) B1,A4,'1987'
```

***

ANSWER: Statement a is incorrect because the initialized value is longer than the data field.

Statement c is incorrect because apostrophes are used to enclose a decimal item (apostrophes are an alphanumeric notation).

Statement d is incorrect because the initialized value is too short; NUM is defined as a 7-digit decimal field.

Statements b and e are correct usage of the initialization specification.

---

17. QUESTION: We want NUMS to be an array of two decimal numbers, of three characters each. The two numbers are to have constant values of 333 and minus 61 respectively. What is the appropriate data specification?

***

ANSWER: NUMS,2D3,333,061

---

18. QUESTION: What information is generated by this data specification?

```
B,D6,000013
```

***

ANSWER: 000013. The initialized value must be the same length as the data size. In this case, B is defined as a six-decimal digit.

---

19. QUESTION: What information is generated by this data specification?

```
TOT,D1,C8
```

***

ANSWER: None, the value is too long (2 characters) and C is not a decimal digit. Non-digit characters, whether or not enclosed in apostrophes, cannot be used for a decimally defined item.

NOTE

In prior versions of DIBOL, a blank data specification field left the original contents of memory in a field. To clear the fields of a record, RECORD C had to be specified. In order to be compatible with existing programs DIBOL treats RECORD or RECORD C identically. For example:

```
RECORD A,C
       NUM,D6
       TOT,D7
       COST,D4,4999
       B1,A7
```

The initialized value of COST would not be cleared, but NUM, TOT, and B1 memory locations would be cleared. The first statement could have been RECORD A.

---

20. QUESTION: Consider the following operation:

DATE,A8,P

When the program is loaded, the computer types

ENTER DATE

and the operator types 07/07/1970. What happens?

***

ANSWER: Since too many characters were typed, it is an error. 07/07/19 is stored in DATE and 70 is stored in the following field.

RECORD,X

The concept of the overlay is a valuable tool in the preservation of computer memory. By means of the overlay, two RECORD statements can describe exactly the same area of computer memory. Whenever there is an X there must be a previously defined RECORD statement without an X. There can be one or more overlays defining the same area. Note the use of X below.

```
RECORD  A
        A1,A5,'DIBOL'
        A2,A8,'SOFTWARE'
        A3,A7
RECORD  B,X
        B1,A5
        B2,A8
        B3,A7'SYSTEM'
```

In this example, the fields in RECORD B occupy the same area of computer memory as the fields in RECORD A.

---

21. QUESTION: What is the value of data labeled B1?

***

ANSWER: B1='DIBOL'

---

22. QUESTION: What is the value of B2?

***

ANSWER: B2='SOFTWARE'

As a general rule, data specifications in overlays should be consistent. Problems may arise if an alphanumeric item is redefined as decimal in an overlay specification. Normally, initialized values are not used in overlays. The overlay (X) block must be equal to or smaller than the last non-overlayed record.

---

23. QUESTION: Is the following correct?

```
RECORD  A
        A1,D10
        A2,A10
RECORD  B,X
        B1,D5
RECORD  C,X
        C1,D3
        C2,D7
        C3,A5
        C4,A5
```

***

ANSWER: Yes. The redefined records, RECORD B and C, are the same size or smaller than the record they redefine, RECORD A.

---

24. QUESTION: Is the following a legal use of the overlay?

```
RECORD  A
        A1,A3,'FUN'
        A2,A5,'LOVER'
RECORD  B,X
        B1,D8
   ***
```

ANSWER: Yes. However, since B1 is decimal, and the data it redefines is alphanumeric, a run time error would occur if B1 were not cleared before being used in a data manipulation statement.

---

25. QUESTION: Is it legal to name a RECORD X?

***

ANSWER: Yes. It is also legal to have RECORD X,X.

A symbol (be it a data name, record name, or a statement label defined in the PROC section) consists of alphanumeric characters, the first of which must be a letter. Only the first six characters are significant. Anything in excess of six characters is ignored. Data names and statement labels must be followed by a comma. A record that is to be used for input or output has a maximum size of 510 characters. Other records have a maximum size of 4094. A decimal field has a maximum length of 15 digits and cannot have a name. Alphanumeric fields have no size restriction other than record size.

---

26. QUESTION: In the following example, determine any errors in data or record names, in their size or their value assignment:

```
a)          RECORD INPUTBUFFER
b)  NAME,A20
c)  POPULATION,D17
d)  A12Y4X,A21,'ELK␣MOUNTAIN,␣WYOMINC
e)  3ABC,A3,'ABC'
f)          RECORD B
g)  TABLE, 100D5
h)  TABLE1, 10D2
i)          RECORD
j)  TEMP,D
k)  WORKTAB, 200D6
l)  TABLE10, 100D10
   ***
```

ANSWER:

a) The record name INPUTBUFFER is more than six characters and is thus recognized as INPUTB. This, however, is not an error.

b) No errors.

c) The field name POPULATION is more than six characters and will be recognized as POPULA. This is not an error. However, it is defined as a Decimal field containing 17 digits -- decimal fields used in arithmetic operations cannot exceed 15 digits and will generate run-time errors.

d) No errors.

e) The field name 3ABC is invalid. It must start with an alphabetic character.

f) No errors.

g) No errors.

h) This record contains 520 characters. It is an error to give a record a name if it contains more than 519 characters.

i) No errors. This record is less than 4096 characters. Since it has no name, it can never be used for input or output.

j) No errors. TEMP is defined as a Decimal field containing one digit.

k) No errors. The field name is recognized as WORKTA.

l) No errors. TABLE10 is recognized as TABLE1.

Many times a programmer will make comments, so that someone else reading his program will know what he is doing. A semi-colon (;) tells the compiler-program that all information following is not to be interpreted as program text, but rather as comments by the programmer. Thus, comments can appear on a program listing, but will not affect the operation of the program. Here is an example of a comment:

```
START     ;THIS PROGRAM READS INDIVIDUAL TRANSACTIONS
RECORD A;THIS IS THE INPUT RECORD BUFFER
        A1,A16;CUSTOMER'S NAME IS STORED HERE
```

The comment is terminated by a carriage-return line feed. The comment following a START or PROC statement is used as a heading for program listings.

---

27. QUESTION: What are the functions of these computer-defined symbols?

```
,X
;
***
```

ANSWER: The X indicates one record of data elements will overlay the previous record (use the same space in computer memory that the previous record was using); multiple overlays of the same record are permitted. The semicolon indicates the beginning of a comment.

If the X option is used in a RECORD statement without a record name, then a comma must follow the word RECORD, i.e., RECORD,X.

DATA SECTION SUMMARY

The data section describes all data used in a program and causes allocation of memory storage. It consists of one or more data records. Each data record section is made up of a RECORD statement followed by one or more data statements.

1. RECORD STATEMENT

   a) Normal Form - RECORD record name, e.g., RECORD INBUF. All uninitialized fields are cleared.

   b) Unnamed Form - A record name may be omitted, e.g., RECORD. All uninitialized fields are cleared.

   c) Data Overlay - Overlay a preceding storage area, e.g., RECORD,X or RECORD B,X. All uninitialized values are cleared.

2. DATA STATEMENT

   a) Normal Form -

   data name, (optional) | data specification, (required) | initialization specif. (optional)

   For example: FIELD1,D4,1234
   FIELD2,A4,'ABC4'

   b) Operator Initialization - Specified by a P and causes entry of data from keyboard before program execution, e.g., RNDATE,A8,P.

   c) Date Initialization - specified by a D and causes entry of the Monitor date.

3. Three ways to initialize data elements in the Data Section:

   Data Statement Initialization - A data statement with an initialization specification. If no value is specified the field is cleared.

   Operator Initialization - A data statement with a ,P which allows entry of data from console.

   Date Initialization - A data statement with a ,D which automatically enters Monitor's date at run time.

PROCEDURE SECTION

28. QUESTION: What is the compiler statement that separates the data and procedure sections?

***

ANSWER: The PROC statement separates the data and procedure sections.

---

29. QUESTION: What is the difference between a procedural statement and a data statement?

***

ANSWER: Procedural statements are executable.

INIT

In a computer program, procedural statements are executed sequentially, the sequential execution of instructions can be changed by a branching instruction.

The first procedural statement discussed is the file-initialization statement. The general form is:

INIT (channel, dev, data file name, logical unit)

The INIT statement is used to associate a channel number with a device and to initialize that device. Channel is a number from 1 to 15 which is to be linked to a logical or physical device. This number is then used in other statements, such as XMIT, to refer to the same device.

Dev is the name of the COS 300 device to be associated with the channel number. These names can be abbreviated, since only the first character is checked. The following list contains the valid dev names:

Dev	Abbreviation	Meaning
IN	I	Mass storage device to be used for input.
OUT	O	Mass storage device to be used for output.
UPDATE	U	Mass storage device to be used for random access.
KBD	K	Input from terminal keyboard.
TTY	T	Output to terminal printer or display.
LPT	L	Line printer.
CDR	C	Card Reader.
RDR	R	Paper tape reader.
PTP	P	Paper tape punch.
SYS	S	Input from a file created on the system device with the editor.

For example:

INIT(1,KBD)

will initialize the terminal keyboard and any references to channel 1 will be references to the terminal keyboard.

Only mass storage devices (disk or DECtape) need be INITed. It is optional for all other devices. If not specified, the following channel number assignments are assumed:

5=PTR
6=LPT
7=KBD
8=TTY

---

30. QUESTION: Write a statement to initialize the terminal display and assign it to channel 8.

***

ANSWER: INIT(8,TTY).

However, COS 300 has already assigned the terminal display to channel 8. This statement is redundant. Only mass storage devices need by INITed.

---

31. QUESTION: Write a statement to initialize the line printer and assign it to channel 1.

***

ANSWER: INIT(1,LPT) or INIT(1,L).
Both statements are identical in DIBOL.

Only mass storage devices specify the data file name, which is required, and a logical unit number, which is optional. The data file name is an alpha constant or a variable which is physically written on this file. It can be up to six characters; anything in excess is ignored. Any valid COS character can be used to make up the name. If a variable is used with the P option, a file name can be specified at run time.

Unit is an optional decimal expression used with I, O and U device codes to specify the logical unit where the data file is stored or to be stored. If the logical unit is not specified, a MOUNT message will occur at run time.

Logical units are specified in SYSGEN and divide the available mass storage into 15 possible areas for data files. These areas can be different sizes (in multiples of 8000 characters) and more than one area can be assigned to one physical device.

---

32. QUESTION: Write the statements necessary to

a) initialize a DECtape data file called MASTER which will be referred to as channel 2 and be used as input.

b) initialize a second DECtape data file called MASTER which will be referred to as channel 3 and be used for output.

c) initialize a card file containing transactions which will be referred to as channel 5 and be used to update the input MASTER file.

***

ANSWER: a) INIT(2,INPUT,'MASTER')
I is sufficient in place of INPUT. Also the following message would occur at run time:

MOUNT MASTER #01 FOR INPUT:

at which time the operator would respond with the logical unit where the file could be found.

b) INIT(3,OUT,'MASTER')
O is sufficient in place of OUT. A message would appear at run time:

MOUNT MASTER #01 FOR OUTPUT:

to which the operator would respond as in (a).

c) INIT(5,CDR)
INIT(5,CDR,'TRANS') would be incorrect since the card reader is not a mass storage device and cannot have a file name associated with it.

---

33. QUESTION: How would you write a statement to initialize a file on logical unit 12 called MASTER which will be accessed directly and which will be referred to as channel 13?

***

ANSWER: INIT(13,UPDATE,'MASTER',12)

---

34. QUESTION: Which of the following INIT statements are invalid and why?

a) INIT(1,INPUT,'FILEA',4)
b) INIT(3,IYBDGHKL,'LABEL')
c) INIT(2,X,'LABEL')
d) INIT(5,KBD)
e) INIT(5,IN'MAST')
f) INIT(8)
g) INIT(13,KEYBOARD)
h) INIT(7,TTY)
i) INIT(15,UPDATE,'.$"#')
j) I(12,K)
k) INIT(1,I,'$TEMP',8)
l) INIT(5,O,TAPEID,3)

***

ANSWER:
c) X is an illegal dev.
e) A comma is missing between IN and MAST.
f) The dev specification is missing.
h) Missing right parenthesis.
j) INIT cannot be abbreviated.

---

35. QUESTION: Statement l in the preceding question is valid since a variable may be used as a data file name. How can the data file name be varied during each run without changing the program?

***

ANSWER: The statement TAPEID,A6,P will allow the operator to enter a six-character file name whenever the program is run.

XMIT

To read or write a record, the transmit data statement is used. Its general form is:

XMIT	(channel,	record,	end of file label)
	(required)	(required)	(only for input file)

Examples of the transmit-data statement are below:

a) XMIT(1,OUTBUF)
b) XMIT(2,INBUF,EOF)

a. Assuming channel 1 has been previously INITialized for output, statement a would transfer the contents of RECORD OUTBUF to channel 1.

b. Assuming that channel 2 has been previously INITialized for input, statement b would transfer data into RECORD INBUF from channel 2.

---

36. QUESTION: What is accomplished by the following DIBOL program?

```
        START
        RECORD INBUF
               INA,A10
               INB,A6
               INC,A6
        RECORD
               DATE,A8,P
```

```
PROC
        INIT(2,IN,'INFILE',14)
BEGIN,XMIT(2,INBUF,EOF)
        GO TO BEGIN
EOF, FINI (2)
STOP
END
```

***

ANSWER: When the program is run, the terminal will output ENTER DATE and wait for the operator to input an eight-character date (note the P option on data-item DATE). Channel 2 will be initialized and all records will be read from channel 2 into the area assigned to RECORD INBUF; after all records are read, the program will transfer to end-of-file routine (EOF) in which channel 2 is rewound, and then the program will stop. The END statement and STOP statement are optional.

NOTE

If the BEGIN statement were BEGIN,XMIT(2,INBUF) an error message is output when an end-of-file occurs.

It is also possible to XMIT partial records. In the previous example assume that the first 10 characters from each input record would be written onto the line printer. The statement would look like:

```
XMIT(6,INBUF(1,12))
```

and would be added after the BEGIN statement. The character count of a record is contained in its first two characters. To output a partial record of 10 characters, the program must specify the first 12 characters. This character count is generated automatically by the COMPiler and does not interfere with the first field in a record.

37. QUESTION: What is the function of the FINI statement?

***

ANSWER: The FINI is a close file statement and must refer to a previously INITialized file. For output files, an end-of-file mark is written onto the file and the file is rewound.

---

38. QUESTION: What is accomplished by statements 1 through 4 in the following example?

```
        START
        RECORD ABC
                A,A10
                NUMS,D15
                BUF,A100
        PROC
1        ,A=
2        ,NUMS=
3        ,BUF(56,70)=
4        ,ABC=
```

ANSWER: Statement 1 sets the ten character field A to spaces.
Statement 2 sets the 15 character field NUMS to zeroes.
Statement 3 sets the characters 56 through 70 of field BUF to spaces.
Statement 4 sets the record ABC to spaces.

An attractive feature of the DIBOL language is the ability to reference characters within a field. The notation BUF (56,70) allows the programmer to reference characters within a data element

without assigning a specific data name. The general format to accomplish this is:

Data name (starting character position, ending character position)

---

39. QUESTION: What would be accomplished by statements 1 and 2 in the following example?

```
    START
    RECORD B
            NUMS, 10D2
            B1,5A6
    PROC
1    ,NUMS (5)=
2    ,B1 (4)=
            ***
```

ANSWER: Statement 1 zeroes the fifth element of the array NUMS.
Statement 2 sets the fourth element in the array B1 to spaces.

This notation is called subscripting. It allows the programmer to reference a specific data element of an array. This form of subscripting must be a positive non-zero number, data name, or expression. The data name option is called variable subscripting. For example:

```
START
RECORD
        NUMS, 10D2
        A,D1
PROC
A=5
NUM(A)=
```

This will accomplish the same as NUM(5)= which is in the previous example.

NOTE

An entire array cannot be referenced, only a single element within an array. However, it is possible to reference an entire array by redefining the array, using RECORD,X (overlay). For example:

```
RECORD
        NUMS,5D2
RECORD,X
        NUMS1,D10
PROC
        S1,NUMS1=
```

This will set the entire array of NUMS to zero.

In summary the Clear Data Statements have the following formats:

```
Destination field = e.g., A=
Destination field (subscript) = e.g., A(4)=
                                or    A(B)=
                                or    A(51,71)=
```

ALPHA=ALPHA

Another type of data manipulation is the move-alphanumeric-variable statement. It takes the general form:

```
alpha field  = alpha field
(destination)   (source)
```

This allows one alpha field to be moved to another alpha field. If the source is shorter than the destination, the result is left-justified with the right-most characters undisturbed. If the source is longer

than the destination, the result is left-justified and the right-most characters are not moved to the destination field.

---

40. QUESTION: What is the value of A in the following example, after the move has been executed?

```
START
RECORD
        A,A5,'ABCDE'
        B,A3,'FGH'
PROC
        A=B
     ***
```

ANSWER: Variable A now has the value FGHDE. The source is shorter than the destination field. The right-most characters are undisturbed.

---

41. QUESTION: What is the value of NAME in the following example?

```
START
RECORD A
        NAME,A4,'FRED'
        NAME1,A7,'JOHNSON'
PROC
NAME=NAME1
     ***
```

ANSWER: NAME now has the value of JOHN.

NOTE

While the receiving field is changed (destination), the sending (source) field remains unchanged, so NAME1 still has the value JOHNSON.

In review, the general format of move alpha to alpha data element is:

Alpha field = alpha field
(destination) (source)

DECIMAL=DECIMAL or EXPRESSION
Statement

Another form of data manipulation is moving a decimal expression to a decimal field. The general format for this expression is:

decimal field = arithmetic expression
(destination) (source)

The arithmetic expression may be any expression with decimal elements, subscripted data elements, constants, and the operators plus (+), minus (-), multiply (*) and divide (/). The contents of parentheses are performed first, division and multiplication next, followed by addition and subtraction. The destination field would be right-justified after the move. Below is an example:

```
     START
     RECORD
             QORDER,D4,0002
             UCOST,D4,0200
             ECOST,D10
             X,D2,04
             Y,5D3,000,007,100,025,023
     PROC
1            ECOST=UCOST*QORDER
2            X=X+1
3            Y(1)=Y(X)+(25*Y(2)+Y(3))/Y(4)
4            X=Y(3)+Y(4)
```

Statement 1) ECOST is calculated by multiplying UCOST and QORDER. In the above example, the answer would be: ECOST=0000000400. The result is right-justified in ECOST with the leading two characters set to zero.

Statement 2) The new value of X shall be X+1 (answer, X=05).

Statement 3) The first element in array Y will be equal to the fifth element in array Y(X=5), plus the following quantity: 25 multiplied by the second element in array Y plus third element in array Y, the result of this multiplication and addition is divided by fourth element in array Y. The answer would be:

Y (1) = Y (05) + (25 * Y(2) + Y(3))/Y(4)

Y (1) = 023 + (25 * 007 + 100)/025

Y (1) = 023 + (275)/25

Y (1) = 023 + 011

Y (1) = 034

Statement 4) X is set equal to 25. If the destination field is too small to contain the source field or source expression, the high order digits are lost.

---

42. QUESTION: In the following expression, explain the items, i.e., decimal field, subscripted field, constants, and the operators, plus, minus, multiply, and divide.

X=Y(3)+Y(2)+66*(13-Z)/2

***

ANSWER: Subscripted variables are Y(3), Y(2); decimal variable is Z; constants are 66, 13, 2; the arithmetic operators used are +, *, -, /.

NOTE

The words variable and field can be used interchangeably.

---

43. QUESTION: Is the expression X=Y(2) equal to X=Y*2?

***

ANSWER: No. Y(2) is a subscripted data element denoting the second element of an array with the name Y. The expression X=Y*2 is the equivalent of multiplying Y times 2 and storing it in X.

---

44. QUESTION: What is the expression which would accomplish the following?

a) Take a number X and add it to the second element in an array named K.

b) Take the result of that operation and divide it by 145 and store it in M.

***

ANSWER: M=(X + K (2))/145
If X + K(2) were not in parentheses, K(2) would be divided by 145 before adding X.

In summary, the decimal to decimal move has the general format of:

decimal field = arithmetic expression

A = A + B/C
A = B

Decimal to Alpha
Alpha to Decimal

The two forms of converting from one data type to another are:

a) Decimal field = alpha field

b) Alpha field = decimal field or decimal expression and an optional format.

---

45. QUESTION: In the following example, data fields are described in both alphanumeric and decimal formats. Convert TOT from decimal to an alpha format of corresponding length and store in A1; and convert NUM to its decimal format of corresponding length and store in B1.

```
START
RECORD A
        NUM,A6
        A1,A6
RECORD B
        TOT,D6
        B1,D6
PROC
        ***
```

ANSWER: A1=TOT (converts decimal to alpha)

B1=NUM (converts alpha to decimal)

The result of the conversion is always stored in the destination field (the expression located to the left of the equal sign). The decimal-to-alpha conversion is always right-justified with leading spaces, if needed. If the destination field is too small, high order characters are lost. The alpha-to-decimal conversion is also right-justified with leading zeroes, if needed. If the destination field is too small, high order characters are lost.

---

46. QUESTION: What would be the contents of B1 and A1 after the following conversions?

```
START
RECORD A
        COST,D4,9999
        A1,D5
RECORD B
        NUM,A6,'678912'
        B1,A6
PROC
        B1=COST
        A1=NUM
        ***
```

ANSWER: B1=COST (converts decimal to alpha). B1 would contain 9999 right-justified with two leading blanks. A1=NUM (converts an alpha-numeric number to decimal). A1 would contain 78912; the high order character is lost.

Decimal to Alpha with Format

In business data processing, it is frequently desirable to output decimal information with imbedded commas, a decimal point and (if needed) a minus sign. For example, -34,259.00 is easier to read than -3425900. DIBOL makes it possible to accomplish the formatting of decimal information during the conversion from decimal-to alpha format. The general form of conversion is:

alpha field=decimal field or decimal expression, format

For example, if B=125677700 (decimally formatted), the expression, A=B, '-X,XXX,XXX.XX' will move B to A and cause A to look like this:

1,256,777.00 with no minus sign, since the number is greater than zero. A must be defined as an A13 to hold a full nine-digit negative number.

---

47. QUESTION: For B=4432567 - (assume two decimal places); what would the conversion instruction look like?

***

ANSWER: A=B'-XX,XXX.XX' or 'XX,XXX.XX-'
(-44,325.67) (44,325.67-)

The minus sign in the edit format can be either on the left or on the right. If the decimal value is positive, the sign will appear as a blank.

---

48. QUESTION: Since commas are inserted only if the corresponding comma has a significant digit to the left, if B=311, what would be the value of A after the following?

```
A=B, '-X,XXX,XXX.XX'
```

***

ANSWER: Where b signifies a space, A would be equal to bbbbbbbbb3.11. When a decimal field is converted, it is right-justified.

---

49. QUESTION: What is output to the terminal by the following program?

```
START
RECORD A
       A1,A7
       A2,A8
       A3,A11
RECORD
       NUM,D6,100000
       B1,A7,'CREDIT ␣'
       B2,A11,'TO DIGITAL'
PROC
       A1=B1
       A3=B2
       A2=NUM,'X,XXX.XX'
       XMIT (8,A)
```

***

ANSWER: CREDIT 1,000.00 TO DIGITAL

Most printing characters on the line printer or terminal can be used in a format string; but the following characters have a special meaning:

X Each X represents a digit and leading zeroes are automatically suppressed.

\- If a minus sign is the first or last character in a format statement, a minus sign is inserted when a number is negative.

. Inserts a period and zeroes are no longer suppressed.

, Inserts a comma if there are significant digits to the left.

Z Suppresses a digit position and right-justifies it.

* If an asterisk is the first character of a format, it replaces all leading zeroes.

Examples:

```
NUM,D3,987
A1,A3
A1=NUM,'XXZ'

result is:  A1=b98    (where b signifies a blank)

NUM,D5,12345
PAY,A9
PAY=NUM,'*X,XXX.XX'

result is:  PAY=***123.45
```

The remaining characters are treated as insertion characters. For example:

```
DATE=102370
A1=DATE,'XX/XX/XX'

result is:  A1=10/23/70

              or

NUM=987
A1=NUM,'XXX0'

result is:  A1=9870
```

When using a comma, period, slash, minus sign, or any other notation, it must be counted as a character position. In the above example using slash, A1 must be defined as an eight-character alphanumeric field.

---

50. QUESTION: In the following example, what is the result of each statement in the PROC division?

```
        START
        RECORD A
                A1,A8
                A2,A4
                A3,A4
                A4,A11
                FMT,A4,'X.XX'
        RECORD B
                DATE,D6,103070
                NUM,D3,123
                COST,D3,999
                TOT,D12,000007894211
        PROC
a)              A1=DATE,'XX/XX/XX'
b)              A2=NUM,'XXZ'
c)              A3=COST,'XXX0'
d)              A4=TOT,'-XXX,XXX.XX*'
e)              A4=TOT,'-XXX,XXX.XX'
f)              A2=NUM,FMT
```

***

ANSWER:

Statement a) A1=10/30/70

Statement b) A2=bb12

Statement c) A3=9990

Statement d) A4=*78,942.11* (an asterisk which is not the first character in a format will act as an insertion character and also replace leading zeroes.)

Statement e) A4=bb78,942.11

Statement f) A2=1.23

In summary, the data manipulation statements have the following formats:

Format	Example
Clear Field=	A=
Alpha Field=Alpha Field	A=B
Decimal Field=Arithmetic Expression	A=B*C/D
Decimal=Alpha	A1=NUM
Alpha=Decimal	B1=TOT
Alpha=Decimal, format	A=B, '-XX,XXX.XX'

Note that subscripting can be used in any data manipulation statement.

In most of the examples in which subscripting was used, it was done by referencing specific elements of an array, i.e., NUM (2)=. It is often desirable to change the value of the subscript. This is done by using a data name for the subscript. For example:

```
START
RECORD C
        C1,10A5
RECORD
        A,D2
        B,A5,'DIBOL'
PROC
        A=1
        C1(A)=B
```

This places the value of DIBOL in the first element of the array C1. If all elements of the array were to be set to the value DIBOL, the procedure section would look like:

```
        PROC
                A=1
BEGIN,          C1(A)=B
                A=A+1
                IF (A.LT.11) GO TO BEGIN
                STOP
        END
```

NOTE

A powerful feature for the data manipulation statements is that record names can be used. For example:

```
START
RECORD AAA
        A1,A80
RECORD BBB
        B1,A80
PROC
        AAA=BBB
        STOP
END
```

Statement AAA=BBB is valid. A record name can be moved to another record name. Record subscripting is also legal. For example:

```
START
RECORD AAA
        A1,A80
RECORD BBB
        B1,A80
PROC
        AAA(1)=AAA(2)
        STOP
        END
```

GO TO

The next type of statement is the GO TO statement.

51. QUESTION: From the previous section (and using Foldout #3), what is the purpose of the basic GO TO statement?

***

ANSWER: This statement causes the program control to branch to the executable statement in the procedure section with the specified label, and has the form:

GO TO label

The label must be a statement label assigned to the statement in the PROCedure section where control is to be transferred. It cannot be a data name. A data name refers to an element which has been defined in the data section.

---

52. QUESTION: Is the following use of GO TO correct?

```
START
RECORD A
        A1,A90
PROC
        INIT (2, 1,'FILEXX')
        XMIT (2,A)
        GO TO START
        FINI (2)
        STOP
END
    ***
```

ANSWER: No. START is not an executable statement. Executable statements are found only in the procedure section of the program.

---

53. QUESTION: Is the following use of GO TO correct?

```
START
RECORD B
        B1,A50
PROC
        INIT (2,1, 'HOHUM', 6)
LOOP,   XMIT (2, B, EOF)
        XMIT (8, B)
        GO TO LOOP
EOF,    FINI (2)
END
    ***
```

ANSWER: Yes, LOOP is a label associated with a statement in the procedure section. LOOP is not a data name.

Another type of GO TO statement is the computed GO TO. It has the form:

GO TO (label 1, label 2, ..., label n), decimal expression

For example:

GO TO (LOOP, RUN, STOPS), KEY

This statement reads "If decimal variable named KEY is equal to 1, then go to LOOP; if it is equal to 2, then go to RUN; and if it is equal to 3, go to STOPS. If the variable KEY is not equal to 1, 2 or 3, control passes to the next statement in sequence. There can be any number of labels in a computed GO TO statement.

---

54. QUESTION: If NUM is equal to 2, what does the following accomplish?

GO TO (X1, X2, X3), NUM

***

ANSWER: The program branches to the statement labeled X2.

---

55. QUESTION: In the above example, if NUM is equal to 6, what happens?

***

ANSWER: Control passes to the next statement in sequence.

IF

An IF statement transfers control on the basis of the results of an expression. The form of the statement is:

IF(expression 1 .rel. expression 2) statement

The data items for comparison may be constants, variables, or arithmetic expressions. They must be both alphanumeric or both decimal. The relations are:

.EQ.	equal
.NE.	not equal
.LT.	less than
.LE.	less than or equal
.GT.	greater than
.GE.	greater than or equal

NOTE

The format requires a period immediately before and after the two character relation codes. If an expression is an alphanumeric constant, it must be enclosed in apostrophes.

The statement is executed if the relation is true. Statement is one of the following:

GO TO LABEL
CALL LABEL
RETURN
ON ERROR
STOP
TRACE
NO TRACE

(Options which are unfamiliar will be explained later in this section.)

---

56. QUESTION: Write an equivalent DIBOL statement for the following. If NUMB is less than or equal to 46, then go to the statement labelled LOOP.

***

ANSWER: IF (NUMB .LE. 46) GO TO LOOP

In a decimal comparison, the shorter of two fields is left zero filled before the comparison.

---

57. QUESTION: Write a DIBOL statement to do the following: If DESC is equal to the alpha constant HAPPY, terminate program execution.

***

ANSWER: IF (DESC .EQ. 'HAPPY') STOP

NOTE

Two fields to be compared may be of unequal length. The longer of two Alpha fields is shortened on the right to the same length as the shorter field.

---

58. QUESTION: Is the following use of the IF statement correct?

```
START
RECORD
        NUMB, D3, 223
        ALPH, A3, 'ZAP'
        TOTL, D3,999

PROC
BEGN,   NUMB = NUMB+1
        IF (NUMB .EQ. ALPH) TRACE

STOP
END
```

***

ANSWER: Use of the IF statement is incorrect. NUMB (which is a decimal item) cannot be compared with ALPH (which is an alpha item). A compiler error will result.

CALL
RETURN

When the same coding is used several times in a program, it may be written once as a subroutine. To use the subroutine write:

CALL label

The CALL statement does two things. It saves the address of the statement following the CALL in the RETURN statement of the subroutine and then performs an unconditional branch to a subroutine. The return from a subroutine is to the next statement after the CALL statement. This is accomplished by the RETURN statement. For example:

```
PROC
CALL LIST
;control returns here
.
.
.
LIST, B=B+1
      RETURN
```

NOTE

Control can pass directly to a subroutine. However, its RETURN statement, when executed, will cause a Run-Time error.

---

59. QUESTION: Is the following correct use of the subroutine?

```
        START
        .
        .
        PROC
        .
        .
        CALL SUB1
        .
        .
SUB1,   X = X + 1
        IF (X .NE. 3) GO TO EXIT
        CALL SUB2

EXIT,   RETURN
SUB2,   X = X * 2
        RETURN
```

***

ANSWER: The example is correct. SUB2 is an example of a nested subroutine, called by SUB1. SUB1 calls SUB2, which multiplies the variable X by 2 and returns to SUB1, which returns to the instruction following the original CALL statement. If a subroutine is entered other than by a CALL it is treated in sequential coding, not as a subroutine.

STOP

STOP causes the program to terminate its execution and to return control to the DIBOL Monitor. For example:

```
START
BLOCK  A
       A1, A10
       A2, A2
RECORD B, X
       B1, A12
PROC
       INIT (2, IN, 'AFILE', 3)
LOOP,  XMIT (2, A, EOF)
       XMIT (8,B)
       GO TO LOOP
EOF,   FINI (2)
       STOP
END
```

This example would print each record from logical unit 3 onto the terminal until end-of-file was reached. At that time, control would transfer to EOF where logical unit 3 would be closed. The program would then terminate by transferring control to the DIBOL Monitor. In this example STOP is optional since it is physically the last statement in the program.

On Foldout #3 is a listing of a complete DIBOL-coded program. Examine it, and answer the following questions.

---

60. QUESTION: From statements 15 and 17, what is the function of INBUF?

***

ANSWER: INBUF is the input record into which all data from channel 1 is stored.

---

61. QUESTION: From statements 16 and 20, what is the function of the block named OUTBUF?

***

ANSWER: It is the output record from which all records are written by channel 2.

---

62. QUESTION: What is the purpose of the X in describing OUTBUF?

***

ANSWER: The five fields of OUTBUF occupy the same area of computer memory as the five fields of INBUF (the fifth field of INBUF is not labelled.

---

63. QUESTION: Which statement separates the data section from the procedure section?

***

ANSWER: The PROC statement.

---

64. QUESTION: Why should the input record occupy the same area of computer memory as the output record?

***

ANSWER: With the exception of the field named ECOST, the output records contain the same information as the input record. Thus, not only is computer memory saved, but many more instructions needed to move fields from one buffer to another are eliminated.

---

65. QUESTION: ECOST is defined as decimal and the field it overlays is alphanumeric. Is this valid?

***

ANSWER: Yes. A decimal field may be defined as alphanumeric (and vice versa).

---

66. QUESTION: Put statement number 18 into your own words.

***

ANSWER: "If data name STOCKN is less than 1000, then go to the instruction labeled LOOP. Otherwise, execute the next sequential instruction."

---

67. QUESTION: In the example program, STOCKN refers to a stock number, DESC refers to an item description, UCOST refers to unit cost of the item, QORDER is the quantity ordered, and ECOST denotes the extended cost. Describe in your words the operation of this program (the logic).

***

ANSWER: The program reads records containing a stock number, item description, unit cost, and quantity ordered. It skips records which have a stock number less than 1000. Output records are generated with the same information as the input with an additional item -- an extended cost which is the product of the unit cost and the quantity ordered.

---

68. QUESTION: Put statement 16 into your own words.

***

ANSWER: INITialize channel 2 as an output device which will write a file called ITEM on logical unit 12.

---

69. QUESTION: There doesn't seem to be any way for the program to execute statements beyond statement 21 (an unconditional branch). How is the statement labelled EOF executed?

***

ANSWER: Statement 17 carries the solution. It says "Read a record from channel 1 and store the information in the record labelled INBUF. If there are no more records, go to the instruction labelled EOF."

---

70. QUESTION: How do we know statement 17 is a read statement? (two reasons)

***

ANSWER: First, statement 15 INITializes channel 1 which contains a file called 'ITEM' as an input device; second, only read uses of the XMIT statement have three parameters (channel, record, and end-of-file routine name); write statements have only two parameters (channel and record).

FOLDOUT #3

SAMPLE PROGRAM #2

```
START                                                    ;1
RECORD  INBUF                                            ;2
        STOCKN,   D4                                     ;3
        DESC,     A25                                    ;4
        UCOST,    D5                                     ;5
        QORDER,   D4                                     ;6
        ,         A9                                     ;7
RECORD  OUTBUF,X                                         ;8
        A1,       D4                                     ;9
        A2,       A25                                    ;10
        A3,       D5                                     ;11
        A4,       D4                                     ;12
        ECOST,    D9                                     ;13
PROC    2                                                ;14
        INIT (1,IN,'ITEM',4)                             ;15
        INIT (2,OUT,'ITEM',12)                           ;16
LOOP,   XMIT (1,INBUF,EOF)                               ;17
        IF (STOCKN.LT.1000) GO TO LOOP                   ;18
        ECOST=UCOST*QORDER                               ;19
        XMIT (2,OUTBUF)                                  ;20
        GO TO LOOP                                       ;21
EOF,    FINI(2)                                          ;22
        FINI(1)                                          ;23
        STOP                                             ;24
END
```

The program reads records containing a stock number, item description, unit cost and quantity ordered. It skips records which have a stock number less than 1000. Output records are generated with the same information as the input with an additional item -- an extended cost which is the product of the unit cost and the quantity ordered.

EXPLANATION OF FOLDOUT #3
INVENTORY PROBLEM

```
START
RECORD  INBUF                   ;input record
        STOCKN,D4               ;Stock number, 4 digits
        DESC,A25                ;Description, 25 characters
        UCOST,D5                ;Unit cost, 5 digits
        QORDER,D4               ;Quantity ordered, 4 digits
        ,A9                     ;Unused field
RECORD  OUTBUF,X                ;A redefinition of the input record
        A1,D4
        A2,A25
        A3,D5
        A4,D4
        ECOST,D9                ;Extended Cost, 9 digits
PROC    2                       ;Begin Procedure Section, 2 mass storage
                                ;files will be open at one time
        INIT(1,IN,'ITEM',4)     ;Initialize file 1 as an Input device
        INIT(2,OUT,'ITEM',12)   ;Initialize file 2, as an Output device
LOOP,   XMIT(1,INBUF,EOF)       ;Read a record from channel 1, and
                                ;store it in record INBUF
        IF(STOCKN.LT.1000)      ;If stock number is less than 1000,
          GO TO LOOP            ;read another record
        ECOST=UCOST*QORDER      ;Extended cost would be calculated
                                ;for STOCKN 1000 or over
        XMIT(2,OUTBUF)          ;Write the record OUTBUF onto
                                ;channel 2
        GO TO LOOP              ;Go to statement LOOP to read
                                ;another record
EOF,    FINI(2)                 ;OUTBUF file is closed and EOF mark
                                ;is put at end of file
        FINI(1)                 ;INBUF file is closed
        STOP
END                             ;OUTBUF file will contain all stock
                                ;items with a stock number of 1000
                                ;or over, with DESC, UCOST,
                                ;QORDER, and ECOST for each item.
```

INCR

The INCR (increment) statement adds ones to a specified field and has the form

INCR decimal field

The next two statements are identical

```
DECFLD=DECFLD+1
INCR DECFLD
```

Refer to statement 25 of Foldout #4 for another example.

---

71. QUESTION: Given the following program which statements are invalid?

```
    START
    RECORD
    A,D4
    B,A5
    C,D3
    PROC
a)  INCR B
b)  INCR A
c)  INCR A+C
d)  C=INCR A
```

***

ANSWER: Statement a is invalid since variable B is defined as alphanumeric.
Statement b is correct.
Statement c is invalid since expressions are not allowed in an INCR statement.
Statement d is invalid since INCR cannot be part of a data manipulation statement.

Look at Foldout #4 and its explanation. This program contains samples of the remaining statements to be explained in this section.

FORMS

The FORMS statement is used to format line printer output. It may not be used with any other output device. It has the form:

FORMS (channel, skip-code)

Channel is the channel number associated with the line printer. The skip-code specifies the action to be taken:

0	go to top of next page (skip to channel 1 of the vertical forms control tape).
1 - 4095	skip this number of lines.
-1	(LS8-E only) skip to channel 2 of the vertical forms control tape.
-2	(LS8-E only) print enlarged characters for the next XMIT statement. Since characters are twice their normal width only 66 characters can be printed.

For example:

FORMS(6,3)

means skip three lines on the line printer;

FORMS(6,0)

means skip to the top of the next page;

FORMS(6,N)

means perform the function specified by the value of N.

---

72. QUESTION: In the following program, what is the result of each FORMS statement?

```
        START
        RECORD
        I,D2,20
        J,D2,03
        PROC
            INIT(3,LPT)
a)          FORMS(3,0)
b)          FORMS(6,0)
c)          FORMS(J,3)
d)          FORMS(J,-3)
e)          FORMS(J*2,I+J-13)
```

***

ANSWER: Statement a will allow the line printer to skip to a new page.
Statement b will also allow the line printer to skip to a new page. The line printer may be referred to as channel 3 since it was INITed as channel 3 and as channel 6 since no other device was INITed as channel 6.
Statement c will skip three lines on the line printer. Variables or decimal expressions are allowed for the channel number of skip-code.
Statement d is invalid. The skip-code is incorrect.
Statement e will skip 10 lines on the line printer.
The value of the first expression, J*2, is 6; the value of the second expression, I+J-13 is 10.

TRACE
NO TRACE

These statements are used to debug a program. They can be inserted anywhere in the PROCedure section. The form of the statement is:

```
TRACE
:
:
NO TRACE
```

When the TRACE statement is executed, program tracing is enabled until the execution of a NO TRACE statement. When enabled each DIBOL statement which is executed causes the following line to be printed on the line printer:

```
AT LINE n
```

where n is the source line number. If the statement is a data manipulation statement, the value stored in the destination field is also printed:

```
AT LINE 200
0003
```

TRACing will not occur unless the program is RUN with the /T option (refer to the System Reference Manual for a more detailed explanation of this option).

Indiscriminate placement of TRACE statements will cause excessive output on the line printer. To use a TRACE statement properly, the problem area in a program should be determined and the TRACE/NO TRACE statements used only in the problem area.

---

73. QUESTION: What output will result from the TRACE statement in the following program?

```
(line numbers)
   0100        START
   0110        RECORD
   0120        ITEM,D5
   0130        HOURS,D2
   0140        SALARY,D5
   0150        WAGES,D7
   0160        PROC
   0170              HOURS=40
   0180              SALARY=300
   0190              TRACE
   0200              WAGES=HOURS*SALARY
   0210              IF(WAGES.EQ.10000)NO TRACE
   0220              HOURS=10
   0230              IF(HOURS.EQ.10)GO TO NEXT
   0240              NO TRACE
   0250   NEXT,WAGES=HOURS*SALARY
   0260              NO TRACE
   0270              HOURS=20
   0280              WAGES=HOURS*SALARY
   0290              STOP
                     ***
```

ANSWER:
```
AT LINE 0200
0012000
AT LINE 0210
AT LINE 0220
10
AT LINE 0230
AT LINE 0250
0003000
AT LINE 0260
```

---

74. QUESTION: Which of the following are valid TRACE/NO TRACE statements?

a) IF(A.GT.B) TRACE
b) CALL TRACE
c) NO TRACE
d) GO TO TRACE

***

ANSWER: Statement a is valid.
Statement b will CALL subroutine TRACE, not enable TRACing.
Statement c is valid.
Statement d will GO TO a statement labelled TRACE, not enable TRACing.

ON ERROR

The ON ERROR statement is often inserted in a source program prior to a statement which, if in error when executed, would cause a return to Monitor. The form of this statement is:

```
ON ERROR label
```

where label is a statement in the PROCedure section where control is to be transferred when an error is encountered. An example of this statement is:

```
        ON ERROR FIX
        DECMAL=ALPHA
        :
        :
FIX,
```

Refer to statement 17 of Foldout #4 for another example. The ON ERROR statement eliminates a return to Monitor for the following conditions:

- division by zero;
- in alpha to decimal conversion, a character other than 0 to 9, plus, minus or space;
- more than 15 digits in a decimal field used in a calculation (the field, of course, would have to be defined as D16, D17, or larger);
- an end of file label was not specified in an XMIT statement and the end of the input file was reached;
- input record was greater than its specified size;
- no file was specified in a RUN command to satisfy an INIT(SYS) statement
- direct access of a record beyond the end of a file.

---

75. QUESTION: Which of the following statements need an ON ERROR statement to precede them to prevent program termination?

Assume the following data section

```
START
RECORD
DECMAL,D20,00000000000000050000
I,D5
ALPHA,A5,'ABCDE'
BETA,A5,'12345'
PHI,A3,'+25'
WORK,D8
J,D5,150
```

a) WORK=J/I
b) WORK=I/J
c) WORK=DECMAL*2
d) I=PHI
e) I=ALPHA
f) I=BETA

***

ANSWER: Statement a needs ON ERROR because of division by zero.
Statement b does not need ON ERROR.
Statement c does not need ON ERROR because DECMAL does not contain a value exceeding 15 digits.
Statement d does not need an ON ERROR since a plus sign is a valid character in alpha to decimal conversion.
Statement e needs an ON ERROR statement since none of its characters are valid in alpha to decimal conversion.
Statement f does not need an ON ERROR statement.

---

76. QUESTION: What happens in Sample Problem #2 if statement 17 is:

```
LOOP,XMIT(1,INBUF)
```

***

ANSWER: The program will run properly until the end of file is reached. At that time, the program will return to Monitor because there is no end of file label in the XMIT instruction. This problem can be avoided by leaving statement 17 in its original form or by preceding statement 17 with an ON ERROR statement.

ACCEPT

The ACCEPT statement stores input from the keyboard in a specified alpha field or record as well as the decimal equivalent of the terminating character. It is used mainly with the DISPLAY statement. It has the form:

ACCEPT(terminating character,alpha field)

where the terminating field is usually defined as a two digit field and the alpha field contains the keyboard input. ACCEPT is often used when certain action is to be taken depending upon the value of the terminating character. The values for the terminating characters can be found in Table 1-1 of the System Reference Manual. An example of this statement is:

```
    .
    .
    .
DECMAL,D2
ALPHA,A10
    .
    .
    .
ACCEPT(DECMAL,ALPHA)
    .
    .
    .
```

Another example of this statement is statement 29 of Foldout #4.

---

77. QUESTION: How would you write a program which will ACCEPT 15 alphanumeric characters? If CTRL/U, a terminating character with a value of 21 is typed, restart ACCEPTance of input. Use TCHAR as the terminating character and KBD to store keyboard input.

***

ANSWER:

```
START
RECORD
TCHAR,D2
KBD,A15
PROC
LOOP,ACCEPT(TCHAR,KBD)
    IF(TCHAR.EQ.21)GO TO LOOP
STOP
END
```

---

78. QUESTION: How can the ACCEPT statement in the preceding problem be modified to ACCEPT only 10 characters? (Show two ways.)

***

ANSWER: LOOP, ACCEPT(TCHAR,KBD(1,10))
or define KBD as an A10 field.

DISPLAY

The DISPLAY statement is used primarily with the VT05 terminal to display a message at a specified location on the screen. Any of 20 rows or lines and 72 columns may be specified. The form of the statement is:

DISPLAY(row, column, field)

where row specifies the line and column specifies the column where field is to be displayed. The field may be a decimal constant, an alpha literal, an alpha variable or a decimal variable. The following decimal constants or decimal variables perform a special function:

- 0 = position the cursor at the row and column specified;
- 1 = clear the scope from the row and column specified to the end of the screen and position the cursor at row and column;
- 2 = clear the scope from the column specified to the end of the line and position the cursor;
- 25 = sound the bell or beep and position the cursor.

Any other decimal codes are meaningless.

---

79. QUESTION: How would the DISPLAY statements be written to do the following?

a) Clear column 8 thru 72 of line 12 on the screen.
b) Clear column 3 thru 72 of line 5 and clear lines 6 thru 20.
c) Display the error message 'NOT NUMERIC' at the beginning of the last line.
d) Ring the terminal bell.
e) Clear column 6 thru 8 of line 20.
f) Display the contents of the alpha field XYZ at line 1, column 1.
g) Move the cursor to row I, column J.

***

ANSWER:

a) DISPLAY(12,8,2)
b) DISPLAY(5,3,1)
c) DISPLAY(20,1,'NOT NUMERIC')
d) DISPLAY(0,0,25) ;row and column could be any value but a 0 value for row will not reposition the cursor.
e) This cannot be done with a DISPLAY command. The minimum that could be cleared is column 6 thru 72 of line 20. However, displaying spaces will work. For example: DISPLAY(20,6,'␣␣␣')
f) DISPLAY(1,1,XYZ)
g) DISPLAY(I,J,0)

---

80. QUESTION: How would you write a program to do the following:

a) Clear the screen.
b) DISPLAY 'CLIENT NUMBER␣'
c) ACCEPT a 5 digit client number.
d) Verify that the digit is numeric when storing it in a five-digit field called TEMP. If incorrect, sound the beep, DISPLAY the error message NOT NUMERIC on the bottom line, wait for the operator to strike any key which indicates he understands the error and reenter the information.
e) Since the number may be less than 5 digits, right justify the number entered before DISPLAYing it.

***

ANSWER:

```
START
RECORD
        TCHAR,D2
        CHAR,A5
        TEMP,D5
        ONE,A1
PROC
LOOP,   DISPLAY(1,1,1)
        DISPLAY(1,1,'CLIENT NUMBER␣')
        ACCEPT(TCHAR,CHAR)
        ON ERROR FIX
        TEMP=CHAR
        CHAR=TEMP
        DISPLAY(1,15,CHAR)
        STOP
FIX,    DISPLAY(0,0,25)
        DISPLAY(20,1,'NOT NUMERIC')
        ACCEPT(TCHAR,ONE)
        GO TO LOOP
```

READ
WRITE

The READ and WRITE statements allow direct access of a specified record. This record may be input from (READ) or output to (WRITE) a specified file. The statement has the form:

```
READ (channel,record,record number)
WRITE (channel,record,record number)
```

where channel is a number from 1 to 15, record is a label previously specified in a RECORD statement, and record number is a constant, variable or arithmetic expression specifying the record number to be read or written. For example:

```
READ(5,RECRDA,20)
WRITE(K,REC,J-4)
```

The first example will READ the 20th record from channel 5 into record RECRDA. The second example will WRITE REC as the J-4th record onto channel K. Refer to Foldout #4 for more examples.

NOTE

For a file to be accessed directly, it must be defined as an UPDATE file. For example:

```
INIT(1,UPDATE,'FILEA',3)
:
READ(1,RECRDA,20)
:
```

---

81. QUESTION: How would you write a program which prints every 10th record on the line printer (assume the records are called RECA, are 50 characters long, are in FILEX in channel 3, logical unit 6, and that direct access will be used.

***

ANSWER:

```
          START
          RECORD RECA
SIZE,     A50
LENGTH,   D5
          PROC
                INIT(3,U,'FILEX',6)
LOOP,     INCR LENGTH
                ON ERROR EOF
                READ(3,RECA,LENGTH*10)
          XMIT(6,RECA)
          GO TO LOOP
EOF,      FINI(3)
          STOP
          END
```

FOLDOUT #4

SAMPLE PROGRAM #3

```
START                                                    ;1
RECORD   ITEM                                            ;2
STOCKN,  D4                                              ;3
DESC,    A25                                             ;4
UCOST,   D5                                              ;5
QORDER,  D4                                              ;6
ECOST,   D9                                              ;7
     RECORD                                              ;8
LINECT,  D2,50                                           ;9
REC,     D5                                              ;10
TCHAR,   D2                                              ;11
     RECORD KBDIN                                        ;12
CHAR     A5                                              ;13
     PROC 1                                              ;14
     INIT(1,UPDATE,'ITEM',12)                            ;15
LOOP, CALL GETKBD                                        ;16
     ON ERROR MESAGE                                     ;17
     REC= CHAR                                           ;18
     NO TRACE                                            ;19
     IF(REC.EQ.100)TRACE                                 ;20
     READ(1,ITEM,REC)                                    ;21
     IF(LINECT.LT.50) GO TO PRINT                        ;22
     FORMS(6,0)                                          ;23
     LINECT =                                            ;24
PRINT,INCR LINECT                                        ;25
     XMIT(6,ITEM)                                        ;26
     GO TO LOOP                                          ;27
MESAGE,DISPLAY(2,1,'NOT NUMERIC')                        ;28
     ACCEPT(TCHAR,KBDIN)                                 ;29
     GO TO LOOP                                          ;30
GETKBD, DISPLAY(1,1,1)                                   ;31
GETA,    KBDIN=                                          ;32
         ACCEPT(TCHAR,KBDIN)                             ;33
         IF(TCHAR.NE.21)RETURN    ;21 = CTRL/U           ;34
DISP     DISPLAY(1,1,2)                                  ;35
         GO TO GETA                                      ;36
```

EXPLANATION OF FOLDOUT #4

DUMP SPECIFIED INVENTORY RECORDS

```
START
RECORD   ITEM              ;Input record
STOCKN,  D4                ;Stock number
DESC,    A25               ;Description
UCOST,   D5                ;Unit cost
QORDER,  D4                ;Quantity on order
ECOST,   D9                ;Extended cost
     RECORD                ;Working storage
LINECT,  D2,50             ;Number of lines printed on current page
REC,     D5                ;Record number of record to be printed
TCHAR,   D2                ;Terminating character in an ACCEPT
                           ;command
     RECORD KBDIN          ;5-character record for reading record no.
CHAR,    A5
PROC 1                     ;A maximum of 1 mass storage device will
                           ;be open at the same time

INIT(1,UPDATE,'ITEM',12)   ;Initialize a file called ITEM found on
                           ;logical unit 12 for direct access
LOOP,    CALL GETKBD       ;Get the record number
     ON ERROR MESAGE       ;Go to MESAGE if the next statement is
                           ;in error (such as CHAR containing non-
                           ;numeric characters)
     REC=CHAR              ;Move CHAR to the numeric field REC
     NO TRACE              ;Disable TRACE mode (it is initially
                           ;disabled)
     IF(REC.EQ.100)TRACE   ;Enable TRACE if record number 100
     READ(1,ITEM,REC)      ;Read record REC from ITEM file
     IF(LINECT.LT.50) GO TO PRINT  ;Skip to new page every 50 lines
     FORMS(6,0)            ;Skip to new page
     LINECT =              ;Clear LINECT
PRINT,INCR LINECT          ;Add 1 to LINECT
     XMIT(6,ITEM)          ;Print specified ITEM record on line
                           ;printer
     GO TO LOOP            ;Get next record number

MESAGE, DISPLAY(2,1,'NOT NUMERIC')  ;Display error message
                           ;terminal
     ACCEPT(TCHAR,KBDIN)   ;Wait for operator response indicating
                           ;he has acknowledged the error message
     GO TO LOOP            ;Get next record number
GETKBD,  DISPLAY(1,1,1)    ;Clear the screen
GETA,    KBDIN=            ;Clear record
     ACCEPT(TCHAR,KBDIN)   ;Accept up to 5 characters; accept is
                           ;terminated when 5 characters are typed
                           ;or when a terminating (non-printing)
                           ;character is typed
IF(TCHAR.NE.21)RETURN      ;If terminating character is not CTRL/U,
                           ;return to main program
DISPLAY(1,1,2)             ;If CTRL/U, clear row 1 and
GO TO GETA                 ;accept input again
```

The program accepts a record number from the keyboard. If the record number is not numeric an error message is displayed and the program waits for the operator to depress a terminating key before restarting. When the record number is numeric, that record is read directly from a file called ITEM and then printed on the line printer.

SECTION SUMMARY

You have completed an in-depth discussion of the DIBOL language. If you do not understand DIBOL clearly, by all means study the section a second time.

In summary, the procedure section has the following instructions:

1) Initialize File Device statement (as input or output).

   General form:

   INIT(channel,dev,data-file-name,unit)

   Example:

   INIT(2,IN,FILEX',3)
   INIT(4,KBD)

2) Transmit statement (Read-from or write-into file).

   General form:

   XMIT(channel,record,end-of-file label for input files)

   Example:

   XMIT(2,INBUF,EOF) Read from
   XMIT(1,OUTBUF) Write into

3) Close File statement.

   General form:

   FINI(channel)

   Example:

   FINI(2)

4) Data Manipulation statement.

   General form:

   destination field=source field or expression

   a) Clear data (destination field =)
   b) Move alphanumeric data (alpha data=alpha data)
   c) Compute decimal data (decimal data= decimal expression)
   d) Convert alpha to decimal (decimal data=alpha data)
   e) Convert decimal to alpha (alpha data=decimal data)
   f) Convert decimal to alpha, formatted (alpha data= decimal data, format)

5) GO TO statement (program control transfers to statement label).

   General form:

   GO TO statement label

   Example:

   GO TO LOOP

6) Computed GO TO statement (program control branches to label1 if the decimal data element is 1, etc.).

   General form:

   GO TO (label1,label2,...,labeln),decimal data element

   Example:

   GO TO (TAX,COST,PRICE),A2

7) IF statement (if the relation between the expressions is true, control goes to statement).

General form:

IF(expression1 .rel.expression2) statement

Examples:

```
IF(A.EQ.B) GO TO C
IF(A.NE.1) TRACE
IF(LINE.GT.50) CALL HEADNG
IF(I.LT.1) RETURN
```

8) Subroutine CALL statement (control goes to statement label).

General form:

CALL statement label

Example:

CALL COST

9) RETURN statement (program control returns to the statement after the last CALL).

General form:

RETURN

10) STOP statement (causes program to terminate and transfers control to the Monitor).

General form:

STOP

11) INCR statement (adds 1 to a specified field).

General form:

INCR decimal field

Example:

INCR DECFLD

12) FORMS statement (formats line printer output)

General form:

FORMS(channel,skip-code)

Examples:

```
FORMS(6,0)
FORMS(6,-2)
FORMS(1,10)
```

13) TRACE statement (enables program tracing for debugging).

General form:

TRACE

14) NO TRACE statement (disables program tracing).

General form:

NO TRACE

15) ON ERROR statement (prevents return to Monitor for certain run time error conditions).

General form:

ON ERROR statement label

Example:

```
ON ERROR EXIT
```

16) ACCEPT statement (used to get input from the terminal when retention of the last character typed is desired).

General form:

ACCEPT(terminating field,alpha field)

Examples:

```
ACCEPT(TCHAR,ALPHA)
ACCEPT(TCHAR,ALPHA(1,10))
```

17) DISPLAY statement (used to put output on the VT05 at a certain row and column).

General form:

DISPLAY(row,column,variable)

Examples:

```
DISPLAY(1,1,'MESSAGE')
DISPLAY(I,J,2)
DISPLAY(I,8,ALPHA)
```

18) READ statement (allows a specified record to be read directly).

General form:

READ(channel,record,record number)

Examples:

```
READ(1,RECNAM,28)
READ(4,RECNAM,REC)
```

19) WRITE statement (allows a specified record to be written directly).

General form:

WRITE(channel,record,record number)

Examples:

```
WRITE(1,RECNAM,33)
WRITE(4,RECNAM,REC)
```

SECTION 3

A Programming Exercise

1. QUESTION: On this page is the definition of a program you are to write. It is imperative you complete writing the program before you look at this author's solution. It is also important that you write the program during the same sitting in which you study the previous two sections, for the simple reason that prompt reinforcement (through application) is the only way to retain the thorough knowledge of DIBOL. Feel free to use Sections 1 and 2 as reference.

YOU ARE TO WRITE A PROGRAM FOR THE ATHLETICS DEPARTMENT OF A COLLEGE. STUDENT RECORDS ARE STORED ON DECTAPE LOGICAL UNIT 15 IN A FILE CALLED 'STUREC', IN THE FOLLOWING FORMAT:

STUDENT NUMBER	LAST NAME	FIRST NAME	CUMULATIVE G.P.A.	SEX	WEIGHT (LBS)	HEIGHT (FEET)

THE COACH WANTS A LIST OF ALL MEN ON CAMPUS WHO HAVE A GRADE POINT AVERAGE ABOVE 85, WHO WEIGH OVER 170.00 POUNDS, AND WHO ARE OVER 5.75 FEET TALL. THE REPORT IS TO LOOK AS SHOWN ON THE NEXT PAGE.

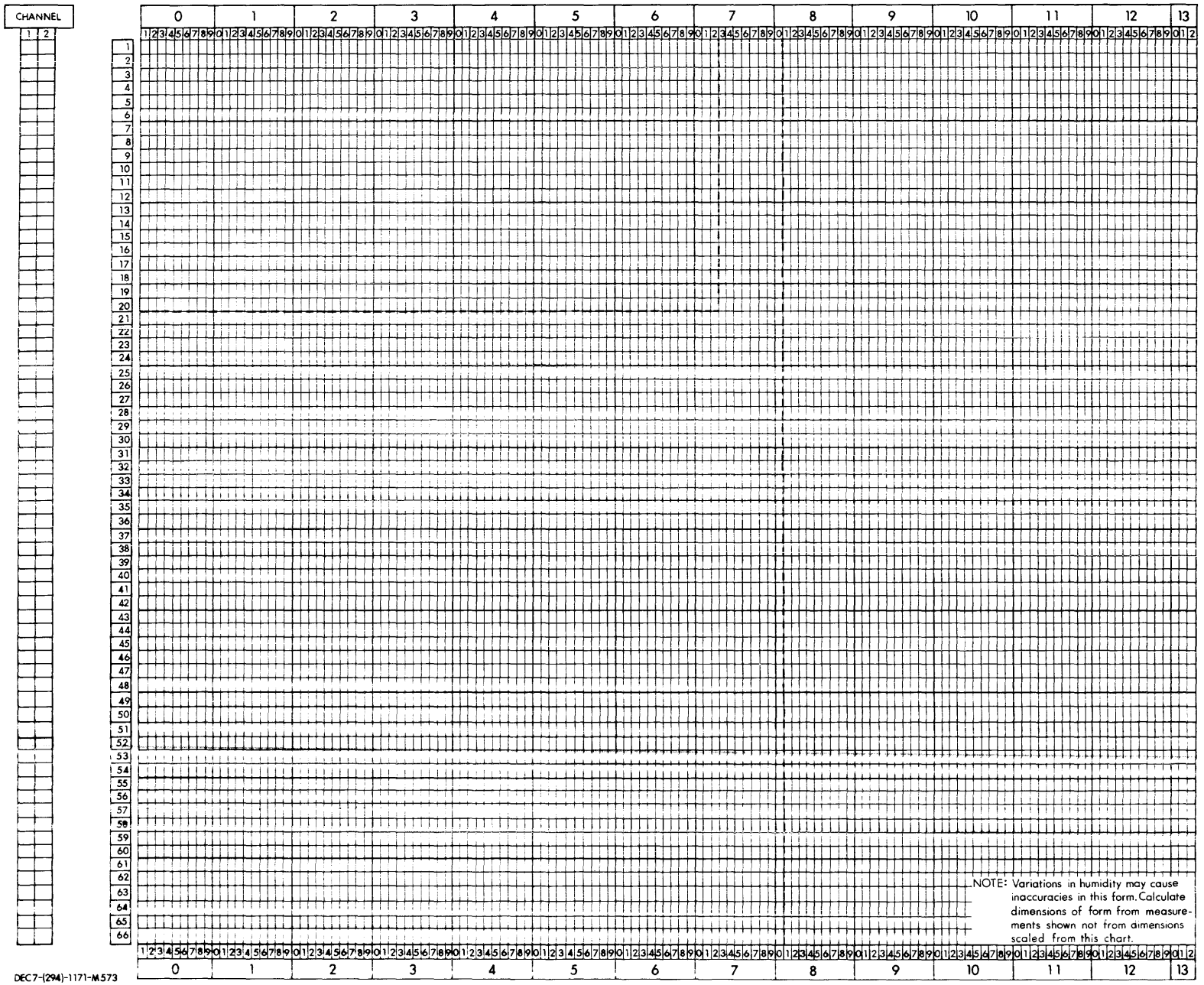
DON'T LOOK AT OUR SOLUTION UNTIL YOU HAVE COMPLETED ALL WORK ON YOURS.

***

digital

decdatasystem

CONSOLE AND PRINTER LAYOUT FORM



DEC7-(294)-1171-M.573

```
START
RECORD  TAPEIN          ;INPUT BUFFER FOR TAPE RECORDS
        STUNO,  D4      ;Student Number
        LNAME,  A10     ;Last Name
        FNAME,  A10     ;First Name
        GPA,    D2      ;Cumulative Grade Point Average
        SEX,    A1      ;Sex (M or F)
        LBS,    D4      ;Weight (XXX.X)
        FEET,   D3      ;Height (X.XX)

RECORD  LPTBUF          ;Line printer Output Buffer
        ,       A3      ;Filler
        LPNO,   A4      ;Student Number
        ,       A9      ;Filler
        LPLNAM, A10     ;Last Name
        ,       A8      ;Filler
        LPFNAM, A10     ;First Name
        ,       A7      ;Filler
        LPGPA,  A2      ;Cumulative GPA
        ,       A9      ;Filler
        LPLBS,  A5      ;Pounds
        ,       A7      ;Filler
        LPFEET, A4      ;Feet

RECORD  HEAD            ;Heading Line of Report
        ,       A37     ;Filler
        ,       A5,'DATE ␣'
        DATE,   A8,P    ;Request date when program is
                          loaded XX/XX/XX
        ,       A18     ;Filler
        ,       A10, 'ATTN:COACH'

RECORD  COL1            ;First line of column heading
        ,       A4      ;Filler
        ,       A3,'STU'
        ,       A19
        ,       A7, 'N A M E'
        ,       A17
        ,       A3, 'CUM'
        ,       A8
        ,       A6, 'WEIGHT'
        ,       A5
        ,       A6, 'HEIGHT'

RECORD  COL2            ;Second line of column heading
        ,       A4      ;Filler
        ,       A3,'NO.'
        ,       A14
        ,       A4, 'LAST'
        ,       A12
        ,       A5,'FIRST'
        ,       A8
        ,       A3, 'GPA'
        ,       A9
        ,       A5,'(LBS)'
        ,       A6
        ,       A4, '(FT)'

RECORD
        LINECT,D2,50
PROC                    ;BEGINNING OF PROCEDURE
                          SECTION
        INIT(1,IN,'STUREC',15)  ;Initialize the input tape
        CALL HEADER     ;Print report headings
REPT,   XMIT(1,TAPEIN,EOF)      ;Read input tape
```

```
        IF (SEX.NE.'M')         ;**Test to determine
          GO TO REPT
        IF (GPA.LE.85)          ;**if record should be
          GO TO REPT
        IF (LBS.LE.1700)        ;**selected.  If a test fails
          GO TO REPT
        IF (FEET.LE.575)        ;**read another record.
          GO TO REPT
        LPNO = STUNO            ;Format print
        LPLNAM = LNAME          ;line by moving
        LPFNAM = FNAME          ;all fields to
        LPGPA = GPA             ;the appropriate print
        LPLBS = LBS,            ;position.  Edit feet
          'XXX.X'
        LPFEET = FEET,          ;and lbs.
          'X.XX'
        CALL PRINT              ;Print the line
        GO TO REPT              ;Read another record

PRINT,  XMIT (6,LPTBUF)         ;Print the line
        INCR LINECT             ;Add one to line count
        CALL HEADER             ;Test if header is to be printed
        RETURN                  ;Return to instruction after last
                                ;call

HEADER, IF (LINECT.LT.50)       ;Print every header after every
          GO TO EXIT            ;50 lines
        LINECT =                ;Set line count to zero
        FORMS(6,0)              ;Skip to new page
        XMIT (6,HEAD)           ;Print Header line
        FORMS(6,1)              ;Print blank line
        XMIT (6,COL1)           ;Print first header line
        XMIT (6,COL2)           ;Print second header line
        FORMS(6,2)              ;Print blank line
EXIT,   RETURN                  ;Return to instruction after last call
EOF,  FINI(1)                   ;Rewind input file
                STOP            ;Return control to DIBOL monitor
        END
```

SECTION 4

Advanced DIBOL Statements

This chapter explains the DIBOL statements which would be used by experienced programmers to:

- Increase system throughput by using print overlap.
- Segment a program which no longer fits in available memory.
- Access source files.
- Do rounding and truncation.

ROUNDING

In DIBOL, all decimal values are stored as integers. It is up to the programmer to keep track of the implied decimal point and to do rounding and truncation.

For example:

```
RECORD
HOURS,D5,04050      ;40.5 HOURS
RATE,D5,02535       ;$2.535 PER HOUR
SALARY,D6           ;IN DOLLARS AND CENTS
PROC
        SALARY=(HOURS*RATE+500)/1000
```

Salary is set equal to 010267 which is actually $102.67. The programmer added 500 to the results of HOURS times RATE in order to round properly. The statement:

```
SALARY=HOURS*RATE/1000
```

would result in SALARY equalling $102.66, which would do truncation without rounding.

An added complication in rounding is the sign of the number. If the result of HOURS times RATE is negative, 500 must be subtracted rather than added. The program to handle positive and negative numbers is:

```
        RECORD
        HOURS,D5,04050-
        RATE,D5,02535
        SALARY,D6
        TEMP,D10
        PROC
            TEMP=HOURS*RATE
            IF(TEMP.LT.0) GO TO TAG1
            SALARY=(TEMP+500)/1000
            GO TO TAG2
TAG1,       SALARY=(TEMP-500)/1000
TAG2,       .
            .
            .
```

All the statements in the procedure section can be replaced by the following:

```
SALARY=(HOURS*RATE)#3
```

The form of this operator is:

```
decimal variable #n
```

where n is a decimal constant or decimal variable in the range of 1 to 7. The decimal variable will be rounded and truncated n places. There is no restriction to the number of # operators in a statement. The # operator has higher priority than all other arithmetic operators; therefore, rounding and truncation are performed before all other arithmetic operations.

The following are some simple examples of # operator usage:

Example	Result
X=1234#1	X=123
X=1234#2	X=12
X=1234#3	X=1
X=5555#3	X=6

---

1. QUESTION: In the previous example, assume that SALARY is three decimal places. What is the statement that stores the product of HOURS and RATE in SALARY?

   ***

   ANSWER: SALARY=(HOURS*RATE)#2

---

2. QUESTION: What is the value stored in SALARY in the following example:

   SALARY=HOURS*RATE#3

   ***

   ANSWER: 012150 or $121.50. RATE is rounded and truncated before being multiplied by HOURS.

---

3. QUESTION: What statements will do the following: round HOURS to the nearest hour and RATE to the nearest dollar, multiply the result, and store in SALARY (assume that SALARY is in dollars)?

   ***

   ANSWER: SALARY=HOURS#1*RATE#3

---

4. QUESTION: What is the value stored in SALARY in the following:

   SALARY=HOURS#1*RATE#3

   ***

   ANSWER: 000120

---

5. QUESTION: What is the value in HOURS and RATE after rounding the following:

   HOURS=2347
   RATE=2347-
   HOURS=HOURS#2
   RATE=RATE#2

   ***

   ANSWER: HOURS contains 23 and RATE contains minus 23.

CHARACTER CONVERSION

The # operator is also used to convert a character to its equivalent internal code and make that decimal number available to the program. When # precedes a variable, it is used to obtain the internal code of the left-most character of the variable. Refer to Appendix A of the COS 300 SYSTEM REFERENCE MANUAL for a table of internal COS codes (COS codes are base 8).

---

6. QUESTION: What is the value of J in the following example if the internal code for 6 is 27 (base 8) or 23 decimal?

```
RECORD
I,D1,6
J,D2
PROC
   J=#I
   ***
```

ANSWER: J contains 23.

---

7. QUESTION: What is the value of J in the following case:

```
   J=I#1
   ***
```

ANSWER: J equals 1. When the number sign follows the variable, it is used for rounding and truncation.

---

8. QUESTION: What is the value of J (the internal code for 7 is 24)?

```
RECORD
I,D2,67
J,D2
PROC
J=#I
   ***
```

ANSWER: J equals 23. The left-most character in the variable is converted.

---

9. QUESTION: What statement will convert the second character in I?

```
   ***
```

ANSWER: J=#I(2,2)

---

10. QUESTION: What is the value of J in the following three statements if the decimal code for A is 34, B is 35, and C is 36?

```
   I,A3,'ABC'
   J,D2
   PROC
1     J=#I(1,1)
2     J=#I(2,2)
3     J=#I(3,3)
   ***
```

ANSWER: In statement 1, J is 34.
In statement 2, J is 35.
In statement 3, J is 36.

SOURCE FILES

The main use of converting the internal code is when processing source files. At run time, the user may specify up to seven source files for input to a DIBOL program. The statement:

```
INIT(n,SYS)
```

will open the first source file. The first record is read with an XMIT statement. The first two characters in that record contain the line number and may be converted to decimal with the following statements:

```
          RECORD SRC
LINE,     A2
TEXT,     A120
          RECORD
LINENO,   D4
          PROC
             :
             :
          LINENO=#LINE*64+#LINE(2,2)
```

---

11. QUESTION: In the procedure section of the above example, fill in the missing statements which would initialize the source file and read the first record.

***

ANSWER:

```
INIT(n,SYS)
XMIT(n,SRC,EOFRTN)
```

When end-of-file is reached on a source file, the file should be FINIed. The next source file must be INITed before being read. The INIT statement can be preceded by an ON ERROR statement which will detect that no files are present. A source file can be processed only once.

---

12. QUESTION: Write a program which will read from one to seven source files. The program will display:

LINE FOUND AT RECORD n IN SOURCE FILE n

if a line number of 3458 was found or will display:

NOT FOUND

if no such line number exists. The message should be printed near the middle of a CRT screen.

***

ANSWER:

```
RECORD    SRC
LINE,     A2
,         A120
RECORD    MSG
,         A21,'LINE FOUND AT RECORD'
RECNO,    D3
,         A16,'IN SOURCE FILE'
SRCNO,    D1

PROC
          DISPLAY(1,1,1)  ;CLEAR SCREEN
BEGIN,    ON ERROR MISSING
          INIT(1,SYS)
          RECNO=
          INCR SRCNO
LOOP,     XMIT(1,SRC,EOF)
          INCR RECNO
          IF (#LINE*64+#LINE(2,2).NE.3458)
             GO TO LOOP
          DISPLAY(10,25,MSG)
          STOP
EOF,      FINI(1)
          GO TO BEGIN
MISSING,  DISPLAY(10,30,'NOT FOUND')
          STOP
```

The Monitor stores tabs in the source file as a single character with an internal code of 61 decimal. When LISTing a program on the line printer or terminal, the Monitor converts tabs into spaces.

---

13. QUESTION: Write a DIBOL program which will detect tabs in a source file.

***

ANSWER:

```
RECORD      SRC
,           A2
TEXT,       120A1
RECORD
I,          D3
PROC
            INIT(1,SYS)
LOOP,       XMIT(1,SRC,EOF)
            I=
LOOP1,      INCR I
            IF (#TEXT(I).EQ.61) GO TO TAB
            IF(I.EQ.120) GO TO LOOP
            GO TO LOOP1
              .
              .
              .
TAB,                    ;FOUND A TAB
              .
              .
              .
```

TRAP

Whenever reports are being printed, the entire computer is tied up doing that task. Much better use of the computer could be obtained if it were allowed to do some other task which does not use the line printer. This is possible in DIBOL with the use of the TRAP statement. Two tasks may be done concurrently. The line printer is given priority and interrupts the other task whenever the line printer is free.

The following program prints numbers 1-500 on the line printer while some other task is being performed:

```
            RECORD A
N,          D3
            PROC
            TRAP SUB
            FORMS(6,0)          ;START LINE PRINTER
              .
              .
              .                 ;PERFORM TASK
              .
              .
LOOP,       IF(N.LT.500) GO TO LOOP   ;WAIT FOR PRINTING
                                      ;TO TERMINATE
            STOP
SUB,        INCR N
            IF(N.GT.500) RETURN
            XMIT(6,A)
            RETURN
```

In the preceding example, the line printer was started with the FORMS statement. Some task was being performed. Since the FORMS statement was preceded by a TRAP statement, the line printer went to the statement specified by TRAP when it was free. A line was XMITed to the line printer and the program then executed a RETURN statement to resume the task. The XMIT statement in the TRAP routine could have been preceded by another TRAP statement if, when the line printer became free, the program were to go to a different statement.

Note that both the task and the report should be completed before the program ends. If the task ends first, it waits in a loop for the printing to complete.

NOTE

For TD8E DECtapes, print overlap will not take place if more than 78 characters are printed during one TRAP subroutine call. For TC08 DECtapes, print overlap will not occur if more than 230 characters are printed.

---

14. QUESTION: Modify the previous example to print 50 lines on one page and then skip to a new page.

***

ANSWER:

```
            RECORD A
N,          D3
            RECORD
LINE,       D2
            PROC
            TRAP SUB
            FORMS(6,0)   ;START LINE PRINTER
            .
            .            ;PERFORM TASK
            .
LOOP,       IF(N.LT.500) GO TO LOOP
            STOP
SUB,        INCR N
            INCR LINE
            IF(LINE.EQ.51) GO TO FORMS
SUB1,       TRAP SUB
            XMIT(6,0)
            RETURN
FORMS,      LINE=
            TRAP SUB1
            FORMS(6,0)
            RETURN
```

---

15. QUESTION: Write a program which will read cards, verify that columns 1-5 are in numerical ascending sequence (ignore out of order cards), and store all 80 columns on a file named CARDS found on logical unit 1. Concurrent with this task, print a report. This report is already in print format in a file named PRINTR on logical unit 2. Print 50 lines per page, a 1-line heading, and two spaces between heading and detail lines. The heading and detail lines are 70 characters each.

***

ANSWER:

```
            RECORD PRINT
,           A70
            RECORD
OLDSEQ,     D5
LINE,       D2
LPFLAG,     D1
            RECORD HDG
,           A70, '...'
            RECORD CARD
SEQ,        D5
,           A75
            PROC 2
            INIT(3,CDR)
            INIT(1,OUTPUT,'CARDS',1)
            INIT(2,INPUT,'PRINTR',2)
            TRAP HEAD1
            FORMS(6,0)
GETCRD,     XMIT(3,CARD,CRDEOF)
            IF(SEQ.LE.OLDSEQ) GO TO GETCRD
                                ;IGNORE CARD
            XMIT(1,CARD)
            OLDSEQ=SEQ
            GO TO GETCRD
CRDEOF,     IF(LPFLAG.EQ.0) GO TO CRDEOF
                                ;LPFLAG=0 IF
              ;OUT OF CARDS BEFORE REPORT DON
            STOP
HEAD,       TRAP HEAD1
            FORMS(6,0)
            RETURN
```

```
HEAD1,    LINE=
          TRAP LPT
          FORMS(6,2)
          RETURN
LPT,      INCR LINE
          IF(LINE.EQ.51) GO TO HEAD
          XMIT(2,PRINT,LPTEOF)
          XMIT(6,PRINT) ;RETURN TO LPT
               ;ON PRINTER DONE IF NO
               ;TRAP IS SPECIFIED.
          RETURN
LPTEOF,   INCR LPFLAG   ;PRINTING DONE
          RETURN
```

CHAINING

In the smallest COS 300 system, programs can be written which require up to 8K of core memory storage. Occasionally, a program is written which exceeds this size and will not run with the available memory. This problem may be overcome by a recently developed feature in DIBOL called CHAINing. A large DIBOL program may be separated into two or more smaller programs which are executed sequentially. Each program is written and compiled separately. These programs are linked together when they are run by saying:

```
.RUN  PROG0+PROG1+...+PROG7
```

The first program uses a CHAIN statement to load the next desired program. Programs that are loaded by a CHAIN statement do not have their data section cleared (unless specifically instructed), thus permitting one program to pass information to another without saving it on a data file.

The format of a CHAIN statement is:

```
CHAIN n
```

where n is a decimal variable in the range 0 to 7 and is the sequence number of the program as specified in the RUN command.

---

16. QUESTION: What can be done when a program is too large to fit in the available memory?

***

ANSWER: The program may be separated into two or more programs.

---

17. QUESTION: How are these smaller programs linked together?

***

ANSWER: The programs are linked together at run time by specifying the program names in the RUN command.

---

18. QUESTION: How can a DIBOL program be loaded from another DIBOL program?

***

ANSWER: By using the CHAIN statement.

The RUN command specifies which programs will be used. For example:

```
.RUN PROG+PROGA+PROGB
```

The CHAIN statement determines which program will be loaded and run next. If, for example, the statement CHAIN 2 were included in PROG, it would terminate execution of PROG, load PROGB, and begin execution of PROGB.

The data section is always cleared in a DIBOL program (except when initial values are specified). However, the data section of any program loaded by a CHAIN statement is not automatically cleared; it will contain the values of the previous program. If some fields are to be cleared, the program must specify:

```
RECORD    ,C
```

where the ",C" means clear all the fields in this record that do not have initial values.

Look at the following programs. In answering the questions, assume that the RUN command is:

```
RUN  PROG+PROGA

             START                ;PROGRAM PROG
             RECORD OUT
OUT1,        A5
             RECORD
I,           D5
NAME,        A4,'FRED'
DUMMY,       A4
             PROC
             INCR I
             IF(I,EQ.10) STOP
             CHAIN 1
             END

             START                ;PROGRAM PROGA
             RECORD LPT
LPT1,        A5
             RECORD
I,           D5
             RECORD,C             ;WORK AREA
J,           D4,0004
K,           A15
             PROC
             LPT1=I
             XMIT(6,LPT)
             CHAIN 0
             END
```

---

19. QUESTION: What do PROG and PROGA do?

***

ANSWER: PROG increments I by 1 and if I does not equal 10, PROG CHAINs to PROGA which prints the value of I and then CHAINs to PROG. When I equals 10, the programs stop.

---

20. QUESTION: Do the data sections have to be the same size in each program?

***

ANSWER: No; the data section is only as large as needed.

---

21. QUESTION: Do the data sections have to match each other either by records or by fields?

***

ANSWER: No; the programmer has complete freedom in assigning the records, fields, and variable names in the data sections. However, the information to be passed from one program to another must be in the same relative location. In the previous programs, for examole, I in PROG could not be passed to I in PROGA if it

were not in the same relative location of the data section.

---

22. QUESTION: What is the first statement executed in PROG? In PROGA?

***

ANSWER: The first statement in the procedure section.

---

23. QUESTION: What are the values of OUT1, I, NAME, DUMMY, J, and K when PROG is run, when PROGA is chained the first time, and when PROG is chained the first time?

***

ANSWER:

	PROG	PROGA	PROG
OUT1	spaces	1	1
I	1	1	2
NAME	FRED	undetermined	FRED
DUMMY	spaces	undetermined	undetermined
J	undetermined	4	undetermined
K	undetermined	spaces	undetermined

File status is lost between CHAIN operations. Data files must be FINIed before transferring to another CHAIN. Files that are used for input or output will present some problems since when they are INITed in the next CHAIN, they will be at the beginning of the file. The easiest solution to this problem is to use the data file as an UPDATE file. When transferring to another CHAIN, FINI the file, save the record number in some common area for use in the next CHAIN or on return to the current CHAIN.

APPENDIX A

INVOICE DATA ENTRY PROGRAM

```
0005  START    ;INVENT - INVOICE DATA ENTRY
0010
0015           RECORD PDET   ;LINE ITEMS FOR LINE PRINTER(LPT)
0020  ,        A4
0025  PINUM,   A7   ;ITEM NUMBER (PART NUMBER)
0030  ,        A5
0035  PDESC,   A24  ;DESCRIPTION
0040  ,        A4
0045  PQO,     A2   ;QUANTITY ORDERED
0050  ,        A2
0055  PUNIT,   A2   ;UNITS
0060  ,        A2
0065  PQS,     A2   ;QUANTITY SHIPPED
0070  ,        A15
0075  PUNITC,  A6   ;UNIT COST
0080  ,        A1
0085  PEXTC,   A9   ;EXTENDED PRICE
0090           BLOCK,X
0095  PDETL,   A80
0100
0105           RECORD PCUST   ;FOR CUSTOMER LINE ON LPT
0110  ,        A8
0115  PCUSTN,  D5   ;CUSTOMER NUMBER
0120  ,        A4
0125  PSALMN,  D2   ;SALESMAN'S NUMBER
0130  ,        A9
0135  PCUSTO,  A5   ;CUSTOMER'S ORDER NUMBER
0140  ,        A20
0145  PDATEO,  A8   ;DATE ORDERED
0150  ,        A4
0155  PDATES,  A8   ;DATE SHIPPED
0160  ,        A4
0165  PSCODE,  D1   ;SHIPPING CODE
0170
0175           RECORD PINCHD   ;FIRST LINE FOR LPT
0180  ,        A48
0185  PINVNO,  A5   ;INVOICE NUMBER
0190  ,        A6
0195  PINVD,   A8   ;DATE OF INVOICE
0200
0205           RECORD PADDRS   ;ADDRESS LINES
0210  ,        A7
0215  PADDR1,  A30  ;LEFT-HAND SIDE
0220  ,        A8
0225  PADDR2,  A30  ;RIGHT-HAND SIDE
0230           BLOCK PADRSA,X
0235  ,        A37
0240
0245  ;CUSTOMER FILE
0250
0255           RECORD CUSTR   ;DATA RECORD
0260  CUSTNO,  D5   ;CUSTOMER NUMBER
0265  CUSTNM,  A30  ;CUSTOMER NAME
0270  CUSTA1,  A25  ;ADDRESS LINE 1
0275  CUSTA2,  A25  ;ADDRESS LINE 2
0280  CUSTZP,  A5   ;ZIP CODE
0285  CUSTSN,  D2   ;SALESMAN CODE
0290  CUSTDC,  D2   ;DISCOUNT %
0295  CUSTTX,  D1   ;TAX %
0300  CUSTSC,  D1   ;SHIP CODE
0305  ,        D10  ;YEAR-TO-DATE TOTAL
                        (NOT USED IN THIS PROGRAM)
0310
0315           RECORD CUSTX   ;INDEX RECORD
0320  CXKEY,   D5   ;KEY (CUSTOMER CODE)
0325  CXPTR,   D2   ;RECORD POINTER
0330
0335  ;PART FILE
0340
0345           RECORD PARTR   ;DATA RECORD
0350  PARTNO,  A7   ;PART NUMBER
```

```
0355  PARTDS,   A30  ;DESCRIPTION
0360  PARTUT,   A2   ;UNIT TYPE (EA, DZ, BX, ETC.)
0365  PARTUC,   D5   ;UNIT COST
0370
0375            RECORD PARTX  ;INDEX RECORD
0380  PXKEY,    A7   ;KEY (PART NUMBER)
0385  PXPTR,    D2   ;POINTER
0390
0395  ;TEMPORARY FILE TO HOLD LINE ITEMS DURING ENTRY
      OF INVOICE
0400
0405            RECORD TEMPR  ;DATE RECORD
0410  TMPITM,   A7   ;ITEM (PART) NUMBER
0415  TMPQO,    D2   ;QTY ORDERED
0420  TMPQS,    D2   ;QTY SHIPPED
0425  TMPUC,    D5   ;UNIT COST
0430  TMPUNT,   A2   ;UNIT
0435  TMPDES,   A30  ;DESCRIPTION
0440
0445  ;FILE TO HOLD TRANSACTIONS
0450
0455            RECORD TRANS
0460  TRCODE,   A1   ;RECORD CODE (H=HEADER, D=DETAIL)
0465  TRCUST,   D7   ;CUSTOMER CODE
0470  TRCORD,   D5   ;CUSTOMER'S ORDER NUMBER
0475  TRDATE,   D6   ;DATE SHIPPED
0480  TRSALN,   D2   ;SALESMAN'S NUMBER
0485
0490            BLOCK,X
0495            A1
0500  TRITM,    A7   ;ITEM (PART) NUMBER
0505  TRQO,     D2   ;QTY ORDERED
0510  TRQS,     D2   ;QTY SHIPPED
0515  TRUC,     D5   ;UNIT COST
0520
0525  ;FOR KEYBOARD INPUT
0530
0535            RECORD KBDREC
0540  KBDIN,    A30
0545            BLOCK,X
0550  KBDCH,    30A1
0555
0560            BLOCK    ;CURSOR CONTROLS
0565  CURSOR,   D2,00    ;TO POSITION CURSOR
0570  EOS,      D2,01    ;TO CLEAR SCREEN
0575  EOL,      D2,02    ;TO CLEAR LINE
0580  BEEP,     D2,25    ;TO BEEP
0585  HOME,     D2,03    ;TO HOME CURSOR
0590
0595            BLOCK    ;EDIT WORDS
0600  ED52,     A9,'XX,XXX.XX'
0605  ED32,     A6,'XXX.XX'
0610  EDATE,    A8,'XX/XX/XX'
0615
0620            BLOCK    ;CONSTANTS
0625  SPC5,     A5
0630  SPC23,    A23
0635  SPC25,    A25
0640  SPC28,    A28
0645  SPC30,    A30
0650
0655            BLOCK    ;SCRATCH
0660  AX1,      A1
0665  AX2,      A2
0670  AX4,      A4
0675  AX6,      A6
0680  AX9,      A9
0685  DX7,      D7
0687  TEMP,     D2
0690
0695            BLOCK    ;WORK
0700  CIN,      D2       ;CURRENT LINE NUMBER
0705  CINX,     D2       ;CIN+5
0710  CINL,     D2       ;LAST LINE NUMBER IN CURRENT
                           INVOICE
0715  COL,      D2       ;EXPECTED COLUMN FOR INPUT
```

```
0720  INVNO,   D5       ;CURRENT INVOICE NUMBER
0725  NOFIND,  A1       ;NON-BLANK IF CUSTOMER OR
                        PART NOT FOUND
0730  DATORD,  A8       ;DATE OF ORDER
0735  TODAY,   A8       ;TODAY'S DATE (MM/DD/YY)
0740  TODAYD,  D6,D     ;TODAY'S DATE (MMDDYY)
0745  I,       D2       ;INDEX
0750  CX,      D3       ;INDEX TO CUSTOMER FILE
0755  PX,      D3       ;INDEX TO PART FILE
0760  TOTPRC,  D7       ;TOTAL INVOICE PRICE
0765  DISCA,   D7       ;DISCOUNT AMOUNT
0770  TAXA,    D7       ;TAX AMOUNT
0775  PAYA,    D7       ;PAY THIS AMOUNT
0780  TCHAR,   D2       ;TERMINATING CHARACTER FOR
                        'ACCEPT'
0785  LINE,    D2       ;LINE COUNT FOR PRINTER
0795
0800            BLOCK    ;ROUGH TABLES FOR INDEX FILES
0805  ITMTAB,   11A7     ;ITEM TABLE
0810  CUSTAB,   11D5     ;CUSTOMER TABLE
0815
0820            BLOCK    ;HEADINGS FOR CRT
0825  ,         A3
0830  ,         A4, 'ITEM'
0835  ,         A5
0840  ,         A11, 'DESCRIPTION'
0845  ,         A14
0850  ,         A3, 'QTY'
0855  ,         A3
0860  ,         A4, 'UNIT'
0865  ,         A3
0870  ,         A3, 'QTY'
0875  ,         A3
0880  ,         A4, 'UNIT'
0885  ,         A7
0890  ,         A5, 'PRICE'
0895  ,         A4
0900  ,         A2, 'NO'
0905  ,         A30
0910  ,         A5, 'ORDER'
0915  ,         A8
0920  ,         A4, 'SHIP'
0925  ,         A3
0930  ,         A4, 'COST'
0935            BLOCK,X
0940  HEAD1,    A72
0945  HEAD2,    A60
1000  PROC6     ;INVENT ----- INVOICE DATA ENTRY
1001
1002  ;* * * * * * * * * * * * * * *
1003  ;SOURCE FILE INVEN2
1004  ;* * * * * * * * * * * * * * *
1005
1010  ;CREATE ROUGH CUSTOMER INDEX
1015
1020            INIT (7, IN, 'CINDEX', 7)
1025            CX=
1030            I=1
1035  RUFCS1,   INCR CX
1040            XMIT (7, CUSTX, RUFCS5)
1045            IF (CX(3,3).NE.1) GO TO RUFCS1    ;IS THIS
                THE 1ST, 11TH, 21ST, ETC RECORD
1050            CUSTAB(I)=CXKEY       ;SAVE EVERY 10TH
                CUSTOMER IN CUSTAB
1055            INCR I
1060            GO TO RUFCS1
1065  RUFCS5,   CUSTAB(I)=99999
1070            FINI (7)
1075
1080  ;CREATE ROUGH PART INDEX
1085
1090            INIT (9, IN, 'PINDEX', 9)
1095            I=1
1100            PX=
1105  RFITM1,   INCR PX
1110            XMIT (9,PARTX,RFITM5)
```

```
1115            IF (PX(3,3).NE.1) GO TO RFITM1   ;IS THIS THE
                1ST, 11TH, 21ST, ETC. RECORD
1120            ITMTAB(I)=PXKEY   ;SAVE EVERY 10TH ITEM IN
                ITMTAB
1125            INCR I
1130            GO TO RFITM1
1135 RFITM5,    ITMTAB(I)=']  ]  ]  ]  ]  ]  ]'
1140            FINI (9)
1145
1150            TODAY=TODAYD,EDATE
1155
1160 ;CREATE EMPTY 'TEMP' FILE
1162 ;(TEMP HOLDS LINE ITEMS DURING ENTRY OF INVOICE)
1165
1170            I=20
1175            INIT (1,OUT,'TEMP',10)
1180 CLEAN,     XMIT (1, TEMPR)
1185            I=I-1
1190            IF (I.GT.00) GO TO CLEAN
1195            FINI (1)
1200
1205 ;OPEN UP ALL FILES
1210
1215            INIT (1,UPDATE,'TEMP',10)
1220            INIT (2,OUT,'ITRANS',11)
1225            INIT (3,KBD)
1230            INIT (4,TTY)
1235            INIT (5,LPT)
1240            INIT (6,IN,'CUSTFL',6)
1245            INIT (7,IN,'CINDEX',7)
1250            INIT (8,IN,'PARTFL',8)
1255            INIT (9,IN,'PINDEX',9)
1260
1265 ;ASK IF FORM IS IN PRINTER
1270
1275 ASK1,      XMIT (4,'INVOICE FORM IN PRINTER?')
1280            XMIT (3,KBDREC)
1285            IF (KBDIN(1,1).NE.'Y') GO TO ASK1
1290
1295 ;PUT OUT THE DUMMY ENTRIES TO TEST FORM POSITION
1300
1305            FORMS (5,0)        ;TOP-OF-FORM
1310            PINVNO='99999'
1315            PINVD='XX/XX/XX'
1320            PCUSTN=9999999
1325            PSALMN=99
1330            PCUSTO='XXXXX'
1335            PDATEO='XX/XX/XX'
1340            PDATES=PDATEO
1345            PSCODE=9
1350            PINUM='XXXXXXX'
1355            PDESC='XXXXXXXXXXXXXXXXXXXXXXXX'
1360            PQO=99
1365            PUNIT='XX'
1370            PQS=99
1375            PUNITC='999.99'
1380            PEXTC='99,999.99'
1385 ASK2,      XMIT (5,PINCHD)
1390            FORMS (5,11)
1395            XMIT (5,PCUST)
1400            FORMS (5,2)
1405            XMIT (5,PDET)
1409            LINE=16
1410            CALL TOPAGE
1415
1420            XMIT (4,"FORM LINED UP?')
1425            XMIT (3,KBDREC)    ;'Y' IS TYPED WHEN
                FORMS ARE LINED UP
1430            IF (KBDIN(1,1).NE.'Y') GO TO ASK2
1435
1440 ;ASK FOR FIRST INVOICE NUMBER
1445
1450 ASK3,      XMIT (4,"WHAT IS FIRST INVOICE NUMBER?')
1455            XMIT (3,KBDREC)
1485            IF (KBDIN(6,30).NE.SPC25) GO TO ASK3
1487            ON ERROR ASK3
```

```
1490              INVNO=KBDIN(1,5)
1495
1500  ;*************************************************
1505  ;START A NEW INVOICE
1510  ;*************************************************
1515
1520  NEWINV,  DISPLAY (1,1,EOS)    ;CLEAR SCREEN
1525
1530              DISPLAY (1,1,'CUSTOMER NO:')
1535              CINX=1
1540              COL=14
1545              CALL GETKBD
1605              IF (KBDIN(6,30).NE.SPC25) GO TO BADCUS
1607              ON ERROR BADCUS
1610              CUSTNO=KBDIN(1,5)
1612              IF (CUSTNO.LT.1) GO TO BADCUS
1615              CALL GETCUS
1620              IF (NOFIND.NE.' ') GO TO NOCUST
1625              TRCUST=CUSTNO
1630
1635  ;GET CUSTOMER'S ORDER NUMBER
1640
1645              DISPLAY (1,21,'CUSTOMER ORDER: ')
1650              COL=37
1655              CALL GETKBD
1660              TRCORD=KBDIN(1,5)
1665
1670  ;GET DATE ORDERED
1675
1680              DISPLAY (1,45,'DATE')
1685              COL=50
1690              CALL GETKBD
1695              DATORD=KBDIN(1,8)
1700
1705  ;GET SALESMAN'S CODE
1710
1715              DISPLAY (1,59,'SALESMAN ')
1720              AX2=CUSTSN
1725              DISPLAY (1,68,AX2)
1730  NINV5A,     COL=70
1735              CALL GETKBD
1740              IF (KBDIN.EQ.SPC30) GO TO NINV5C
1745
1750              CALL GETD2A
1755              IF (KBDIN(3,30).NE.SPC28) GO TO BADSN
1760              CUSTSN=KBDIN(1,2)
1762              IF (CUSTSN.LT.1) GO TO BADSN
1765              AX2=CUSTSN
1770              DISPLAY (1,68,AX2)
1775              DISPLAY (1,70,EOL)
1780  NINV5C,     TRSALN=CUSTSN
1785
1790  ;ALL IS WELL WITH INITIAL LINE -- PUT OUT HEADINGS
1795
1800              DISPLAY (3,1,HEAD1)
1805              DISPLAY (4,1,HEAD2)
1810              CINL=
1815              TOTPRC=
1820
1825  ;*************************************************
1830  ;START NEW LINE
1835  ;*************************************************
1840
1842  ;GET ITEM NUMBER
1843  ;
1845  NINV7,      INCR CINL
1850              CIN=CINL
1855  NINV8,      AX2=CIN
1860              CINX=CIN+5
1865              DISPLAY(CINX,1AX2)
1870              DISPLAY (CINX,3,EOL)
1875              DISPLAY (CINX,4,CURSOR)
1880              COL=4
1885              CALL GETKBD
1890              IF (KBDIN(8,30).NE.SPC23) GO TO BADINO
1895              IF (KBDIN(1,7).EQ. 'TOTAL ') GO TO TOTALI
```

```
1900              IF (KBDIN(1,1).EQ.'-') GO TO CORECT
1905              IF (KBDIN(1,7).EQ.'RESTART') GO TO NEWINV
1910              IF (CINL.GT.8) GO TO TUMANY
1915              TMPITM=KBDIN(1,7)
1920              PARTNO=TMPITM
1925
1930              CALL GETITM
1935              IF (NOFIND.NE.' ') GO TO NOPART
1940              DISPLAY (CINX,13,PARTDS)
1945              TMPDES=PARTDS
1950
1955  ;GET QTY ORDERED
1960
1965  NINV9,      DISPLAY (CINX,38,CURSOR)
1970              COL=38
1975              CALL GETD2
1980              IF (KBDIN(1,2).EQ.' ') GO TO BADQO
1985              TMPQO=KBDIN(1,2)
1990              AX2=TMPQO
1995              DISPLAY (CINX,38,AX2)
2000
2005  ;DISPLAY UNITS
2010
2015              DISPLAY (CINX,45,PARTUT)
2020              TMPUNT=PARTUT
2025
2030
2035  ;GET QTY SHIPPED
2040  NINV10,     DISPLAY (CINX,51,CURSOR)
2045              COL=51
2050              CALL GETD2
2055              IF (KBDIN(1,2).EQ.' ') GO TO BADQS
2060              TMPQS=KBDIN(1,2)
2065              AX2=TMPQS
2070              DISPLAY (CINX,51,AX2)
2075
2080              AX6=PARTUC,ED32
2085              DISPLAY (CINX,55,AX6)
2090              TMPUC=PARTUC
2095
2100              AX9=PARTUC+TMPQS,ED52
2105              DISPLAY (CINX,64,AX9)
2110
2115  ;INPUT OF LINE COMPLETE
2120
2125              WRITE (1,TEMPR,CIN)
2135              TOTPRC=PARTUC*TMPQS+TOTPRC
2145              GO TO NINV7
2150
2155  ;TOTAL OUT THE INVOICE
2160
2165  TOTALI, CINL=CINL-1
2170              DISPLAY (CINX,1,EOL)
2175              DISPLAY (15,47,'TOTAL PRICE')
2180              AX9=TOTPRC,ED52
2185              DISPLAY (15,60,AX9)
2190
2192  ;COMPUTE & DISPLAY DISCOUNT, IF ANY
2193
2195              IF (CUSTDC.EQ.00) GO TO TOTAL3
2200              DISPLAY (16,47,'DISCOUNT')
2205              AX2=CUSTDC
2210              DISPLAY (16,56,AX2)
2215              DISPLAY (16,58,'%')
2220              DISCA=(TOTPRC*CUSTDC+50)/100
2225              AX9=DISCA,ED52
2230              DISPLAY (16,60,AX9)
2235              DX7=TOTPRC-DISCA
2240              DISPLAY (16,60,AX9)
2245              GO TO TOTAL4
2250
2255  TOTAL3,     DX7=TOTPRC
2260              DISCA=
2265
2267  ;COMPUTE & DISPLAY TAX, IF ANY
2268
```

```
2270  TOTAL4,   DISPLAY (17,47,'TAX')
2275            IF (CUSTTX.EQ.0) GO TO TOTAL5
2280            AX1=CUSTTX
2285            DISPLAY (17,51,AX1)
2290            DISPLAY (15,52,'%')
2295            TAXA=(DX7*CUSTTX+50)/100
2300            AX9=TAXA,ED52
2305            DISPLAY (17,60,AX9)
2310            PAYA=DX7+TAXA
2315            GO TO TOTAL6
2320
2325  TOTAL5,   TAXA=
2330            PAYA=DX7
2335
2340  TOTAL6,   DISPLAY (19,44,'PAY THIS AMOUNT')
2345            AX9=PAYA,ED52
2350            DISPLAY (19,60,AX9)
2600  ;* * * * * * * * * * * * * * *
2601  ;SOURCE FILE INVEN3
2602  ;* * * * * * * * * * * * * * *
2603
2604  ;PRINT OUT THE INVOICE
2605
2610            PINVNO=INVNO
2615            PINVD=TODAY
2620            XMIT (5,PINCHD)
2625
2630            FORMS (5,3)
2635
2637  ;PRINT NAME AND ADDRESS
2638
2640            PADDR1=CUSTNM
2645            CALL PRNTIF
2650            PADDR1=CUSTA1
2655            CALL PRNTIF
2660            PADDR1=CUSTA2
2665            CALL PRNTIF
2670            PADDR1(1,19)=
2675            PADDR1(20,24)=CUSTZP
2680            CALL PRNTIF
2685
2690            FORMS (5,4)
2695
2697  ;PRINT GENERAL INFORMATION LINE
2698
2700            PCUSTN=CUSTNO
2705            PSALMN=CUSTSN
2710            PCUSTO=TRCORD
2715            PDATEO=DATORD
2720            PDATES=TODAY
2725            PSCODE=CUSTSC
2730            XMIT (5,PCUST)
2735            FORMS (5,2)
2740
2745            TRCODE='H'
2750            TRDATE=TODAYD
2755            XMIT (2,TRANS)
2760            TRCODE='D'
2765
2770            I=1
2775            LINE=15
2776            PDETL=
2777
2778  ;PRINT INVOICE ITEMS
2779
2780  PLOOP,    READ (1,TEMPR,I)
2785            PINUM=TMPITM
2790            PDESC=TMPDES
2795            PQO=TMPQO
2800            PQS=TMPQS
2805            PUNIT=TMPUNT
2810            PUNITC=TMPUC,ED32
2815            PEXTC=TMPUC*TMPQS,ED52
2820            XMIT (5,PDET)
2825            INCR LINE
2830
```

```
2835                TRITM=TMPITM
2840                TRQO=TMPQO
2845                TRQS=TMPQS
2850                TRUC=TMPUC
2855                XMIT (2,TRANS)
2860
2865                I=I+1
2866                IF (I.LE.CINL) GO TO PLOOP
2880                CALL BOTPAG
2885
2890  ;TOTAL PRICE LINE
2895
2900                PDETL=
2905                PDETL (59,70)='TOTAL PRICE'
2910                PEXTC=TOTPRC,ED52
2915                XMIT (5,PDET)
2920                INCR LINE
2925
2927  ;CUSTOMER DISCOUNT LINE
2928
2930                IF (CUSTDC .EQ.00) GO TO PLOOP2
2935                PDETL(59,70)='DISCOUNT XX%'
2940                PDETL(68,69)=CUSTDC
2945                PEXTC=DISCA,ED52
2950                XMIT (5,PDET)
2955                INCR LINE
2960
2962  ;CUSTOMER TAX LINE
2963
2965  PLOOP2,       IF (CUSTTX .EQ.0) GO TO PLOOP3
2970                PDETL(59,70)='TAX X%
2975                PDETL(63,63)=CUSTTX
2980                PEXTC=TAXA,ED52
2985                XMIT (5,PDET)
2990                INCR LINE
2995
3000  PLOOP3,       FORMS (5,1)
3005
```

```
3007  ;CUSTOMER NET PRICE LINE
3008
3009                PDET=
3010                PDETL(21,35)='PAY THIS AMOUNT'
3011                FORMS(5,-2)      ;DOUBLE WIDTH CHARACTERS
3012                XMIT (5,PDET)
3013                PDET=
3015                PEXTC=PAYA,ED52
3020                XMIT (5,PDET)
3025                LINE=LINE+3
3030                CALL TOPAGE
3035
3040  ;WAIT FOR OPERATOR
3045                DISPLAY (20,1,0)
3050                CALL GETKBD
3055                INCR INVNO
3060                IF (KBDIN(1,4) .NE.'END') GO TO NEWINV
3065
3070                FINI (1)
3075                FINI (2)
3080                FINI (3)
3085                FINI (4)
3090                FINI (5)
3095                FINI (6)
3100                FINI (7)
3105                FINI (8)
3110                FINI (9)
3115                STOP
3120
3125  ;GET READY TO CORRECT A LINE
3130
3135  CORECT, CINL=CINL-1
3140                KBDIN(1,29)=KBDIN(2,30)
3145                CALL GETD2A
3150                IF (KBDIN(1,2).EQ. ' ') GO TO CORCT2
3155                CIN=KBDIN(1,2)
3160                IF (CIN .EQ.0) GO TO CORCT2
3165                IF (CIN .GT.CINL) GO TO CORCT4
```

```
3170            DISPLAY (CINX,1,EOL)
3175            READ (1,TEMPR,CIN)
3180            TOTPRC=TOTPRC-TMPUC*TMPQS
3185            GO TO NINV8
3190
3195  CORCT2,   DISPLAY (20,1,BEEP)
3200            DISPLAY (20,1,'BAD LINE NUMBER')
3205  CORCT3,   CALL ERAW8
3210            GO TO NINV7
3215
3220  CORCT4,   DISPLAY (20,1,BEEP)
3225            DISPLAY (20,1,'LINE NUMBER TOO BIG')
3230            GO TO CORCT3
3235
3300  ;* * * * * * * * * * * * * * *
3301  ;SOURCE FILE INVEN4
3302  ;* * * * * * * * * * * * * * *
3303
3304  ;GO TO TOP OF NEXT PAGE ON THE FORM
3310  TOPAGE, FORMS (5,42-LINE)
3311            LINE=
3315            RETURN
3320
3325  ;GET TO BOTTOM OF PAGE
3330
3335  BOTPAG,   I=15-I
3336            IF (I .LE.0) RETURN
3340            FORMS (5,1)
3345            LINE=LINE+1
3350            RETURN
3355
3360  ;GET AN ITEM FROM THE KEYBOARD
3362  ;(IF TCHAR IS CTRL/U (21), CLEAR INPUTTED ITEM &
      ACCEPT IT AGAIN
3365
3370  GETKBD, KBDIN=
3375            ACCEPT (TCHAR,KBDIN)
3380            IF (TCHAR .NE.21) RETURN
3385            DISPLAY (CINX, COL,EOL)
3390            GO TO GETKBD
3395            RETURN
3400
3405  ;GET A ONE OR TWO DIGIT NUMBER FROM KEYBOARD
3410
3415  GETD2,    CALL GETKBD
3420  GETD2A,   IF (KBDIN(3,30) .NE.SPC28) GO TO GETD2X
3425            ON ERROR GETD2X
3430            TEMP=KBDIN(1,2)
3435            IF (TEMP.LT.0) GO TO GETD2X
3465            RETURN
3470  GETD2X,   KBDIN=
3475            RETURN
3480
3485  ;GET A CUSTOMER RECORD
3490
3495  GETCUS,   CX=2
3497
3498  ;FIND ROUGH INDEX (WITHIN 10)
3500  GTCUS1,   IF (CUSTAB (CX) .GT.CUSTNO) GO TO GTCUS2
3505            INCR CX
3510            GO TO GTCUS1
3512
3513  ;GET EXACT INDEX
3515  GTCUS2,   CX=(CX-2)*10
3520  GTCUS3,   INCR CX
3525            READ (7,CUSTX,CX)
3530            IF (CXKEY .EQ.CUSTNO) GO TO GTCUS5
3535            IF (CXKEY .LT.CUSTNO) GO TO GTCUS3
3540            NOFIND='X'   ;INVALID CUSTOMER NO
3545            RETURN
3550  GTCUS5,   CX=CXPTR     ;MATCH ON CUSTOMER NO
3555            READ (6,CUSTR,CX)
3560            NOFIND=
3565            RETURN
3570
3575  ;GET A PART RECORD
```

```
3580
3585  GETITM,   PX=2
3587
3588  ;FIND ROUGH INDEX (WITHIN 10)
3590  GTITM1,   IF (ITMTAB (PX).GT.PARTNO) GO TO GTITM2
3595            INCR PX
3600            GO TO GTITM1
3605  GTITM2,   PX=(PX-2)*10
3610  GTITM3,   INCR PX
3615            READ (9,PARTX,PX)
3620            IF (PXKEY .EQ.PARTNO) GO TO GTITM5
3625            IF (PXKEY .LT.PARTNO) GO TO GTITM3
3630            NOFIND='X'    ;INVALID PART NO
3635            RETURN
3640  GTITM5,   PX=PXPTR      ;MATCH ON CUSTOMER NO
3645            READ (8,PARTR,PX)
3650            NOFIND=
3655            RETURN
3660
3665
3670  ;PRINT ONE OR TWO ADDRESS LINES
3675
3680  PRNTIF,   IF (PADDR2 .EQ.SPC30) GO TO PRNTF1
3685            XMIT (5,PADDRS)
3690            RETURN
3695  PRNTF1,   IF (PADDR1 .EQ.SPC30) GO TO PRNTF2
3700            XMIT (5,PADRSA)
3705            RETURN
3710  PRNTF2,   FORMS (5,1)
3715            RETURN
3720
3725  ;*************************************************
3730  ;ERROR ROUTINES
3735  ;*************************************************
3740  ;BAD ITEM NUMBER
3745
3750  BADINO,   DISPLAY (20,1,BEEP)
3755            DISPLAY (20,1,'TOO MANY CHARACTERS')
3760            CALL ERAW8
3765            GO TO NINV8
3770
3775  ;BAD CUSTOMER NUMBER
3780
3785  BADCUS,   DISPLAY (20,1,BEEP)
3790            DISPLAY (20,1,'MUST BE 5-DIGIT NUMERIC')
3795            CALL ERAW8
3800            GO TO NEWINV
3805
3810  ;CUSTOMER NOT FOUND ON FILE
3815
3820  NOCUST,   DISPLAY (20,1,BEEP)
3825            DISPLAY (20,1,'CUSTOMER NOT FOUND')
3830            CALL ERAW8
3835            GO TO NEWINV
3840
3845  ;PART NOT FOUND ON FILE
3850
3855  NOPART,   DISPLAY (20,1,BEEP)
3860            DISPLAY (20,1,'PART NOT FOUND')
3865            CALL ERAW8
3870            GO TO NINV8
3875
3880  ;BAD QTY SHIPPED
3885
3890  BADQS,    DISPLAY (20,1,BEEP)
3895            DISPLAY (20,1,'MUST BE ONE OR TWO DIGITS')
3900            CALL ERAW8
3905            DISPLAY (CINX,COL,EOL)
3910            GO TO NINV10
3915
3920  ;BAD QTY ORDERED
3925
3930  BADQO     DISPLAY (20,1,BEEP)
3935            DISPLAY (20,1,'MUST BE ONE OR TWO DIGITS')
3940            CALL ERAW8
3945            DISPLAY (CINX,COL,EOL)
```

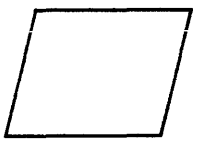
```
3950                GO TO NINV9
3955
3960    ;BAD SALESMAN'S NUMBER
3965
3970    BADSN,      DISPLAY (20,1,BEEP)
3975                DISPLAY (20,1,'BAD SALESMAN NUMBER')
3980                CALL ERAW8
3985                DISPLAY (1,70,EOL)
3990                GO TO NINV5A
3995    ;BAD SALESMAN'S NUMBER
4000    ;TOO MANY ITEMS
4005
4010    TUMANY,     DISPLAY (20,1,BEEP)
4015                DISPLAY (20,1,'TOO MANY LINES')
4020                CALL ERAW8
4025                GO TO NINV8
4030
4035    ;WAIT FOR A CHARACTER TO BE TYPED
4040
4045    ERAW8,      CALL GETKBD
4050                DISPLAY (20,1,EOL)
4055                RETURN
4060
4065    END/N
```

APPENDIX B

STANDARD FLOWCHART SYMBOLS

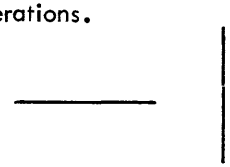
SYMBOLS

1. Input/Output Symbol - Represents an input/output function (I/O), that is, the making available of information for processing (input), or the recording of processed information (output).



2. Process Symbol - Represents any kind of processing function; for example, the process of executing a defined operation or group of operations resulting in a change in value, form or location of information.

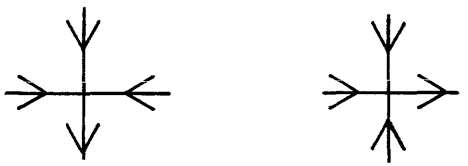
3. Flowline Symbol - Represents the function of linking symbols. It indicates the sequence of available information and executable operations.



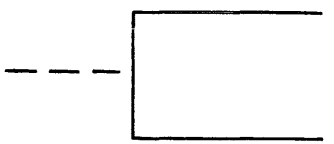
Crossing of Flowlines - Flowlines may cross; this means they have no logical interrelation. Example:

Junction of Flowlines - Two or more incoming flowlines may join with one outgoing flowline. Example:

Every flowline entering and leaving a junction should have arrowheads near the junction point. Example:



4. Comment, Annotation Symbol - Represents the annotation function, that is, the addition of descriptive comments or explanatory notes as clarification. The broken line is connected to any symbol at a point where the annotation is meaningful by extending the broken line in whatever fashion is appropriate.



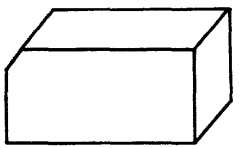
SPECIALIZED SYMBOLS

Specialized I/O Symbols may represent the I/O function and, in addition, denote the medium on which the information is recorded or the manner of handling the information or both.

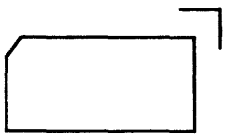
1. Punched Card Symbol - Represents an I/O function in which the medium is punched cards.

The following symbols may be used to represent a deck of cards or a file or cards.

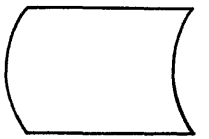
Deck of Cards Symbol. The symbol shown below represents a collection of punched cards.



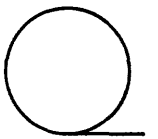
File of Cards Symbol. The symbol shown below represents a collection of related punched card records.



2. Online Storage Symbol - Represents an I/O function utilizing any type of online storage; for example, magnetic tape, magnetic drum, magnetic disk.

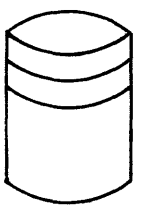


3. Magnetic Tape Symbol - Represents an I/O function in which the medium is magnetic tape.

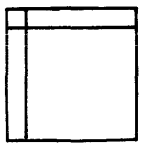


4. Punched Tape Symbol - Represents an I/O function in which the medium is punched paper tape.

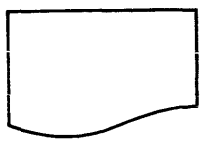
5. Magnetic Disk Symbol - Represents an I/O function in which the medium is magnetic disk.



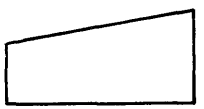
6. Core Symbol - Represents an I/O function in which the medium is magnetic core.



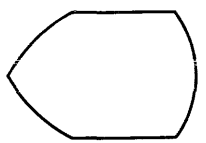
7. Document Symbol - Represents an I/O function in which the medium is a printed document.



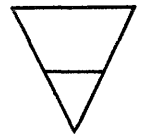
8. Manual Input Symbol - Represents an input function in which the information is entered manually at the time of processing; for example, by means of online keyboards, switch settings, push buttons.



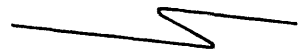
9. Display Symbol - Represents an I/O function in which the information is displayed for human use at the time of processing, by means of online indicators, video devices, console printers, plotters, and so forth.



10. Offline Storage Symbol - Represents the function of storing information offline, regardless of the medium on which the information is recorded.



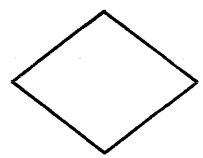
11. Communication Link Symbol - Represents information transmitted by a telecommunication link.



SPECIALIZED PROCESS SYMBOLS

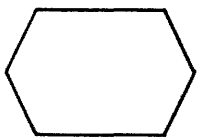
Specialized process symbols may represent the processing function and, in addition, identify the specific type of operation to be performed on the information.

1. Decision Symbol - Represents a decision or switching-type operation that determines which of a number of alternative paths is to be followed.

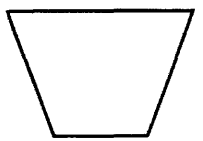


2. Predefined Process Symbol - Represents a named process consisting of one or more operations or program steps that are specified elsewhere; for example, subroutine or logical unit. Elsewhere means not this set of flowcharts.

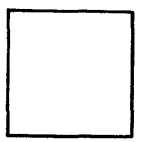
3. Preparation Symbol - Represents modification of an instruction or group of instructions which change the program itself; for example, set a switch, modify an index register, or initialize a routine.



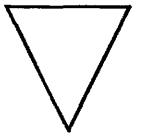
4. Manual Operation Symbol - Represents any offline process geared to the speed of a human being, without using mechanical aid.



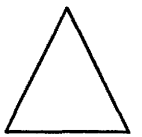
5. Auxiliary Operation Symbol - Represents an offline operation performed on equipment not under direct control of the central processing unit.



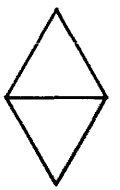
6. Merge Symbol - Represents the combining of two or more sets of items into one set.



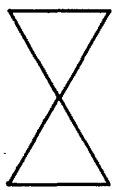
7. Extract Symbol - Represents the removal of one or more specific sets of items from a single set of items.



8. Sort Symbol - Represents the arranging of a set of items into a particular sequence.



9. Collate Symbol - Represents merging with extracting, that is, the formation of two or more sets of items from two or more other sets.

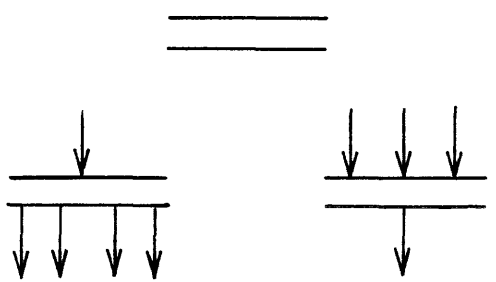


ADDITIONAL SYMBOLS

1. Connector Symbol - The symbol shown below represents an exit to or an entry from another part of the flowchart. It is a junction in a line of flow. A set of two connectors is used to represent a continued flow direction when the flow is broken by any limitation of the flowchart. A set of two or more connectors is used to represent the junction of several flowlines with one flowline, or the junction of one flowline with one of several alternate flowlines.

2. Terminal Interrupt Symbol - Represents a terminal point in a flowchart, for example, start, stop, halt, delay, or interrupt.

3. Parallel Mode Symbol - Represents the beginning or end of two or more simultaneous operations.



GLOSSARY

COS 300 Glossary of Standard Terminology

access time - The time interval between the instant at which data is called from storage, and the instant delivery begins.

alphanumeric - A character set that contains letters, digits, and other characters such as punctuation marks. The alphanumeric character set includes the upper case letters A-Z, the digits 0-9, and most of the special characters on the terminal keyboard. Two of these characters, back slash \ and back arrow ←, are illegal in user data fields.

algorithm - A prescribed set of well-defined rules or processes for the solution of a problem.

analysis - The methodical investigation of a problem, and the separation of the problem into smaller relocated units for further detailed study.

annotation - An added descriptive comment or explanatory note.

array - A DIBOL technique for specifying more than one field of the same length and type. 5D3 reserves space for five decimal fields, each to be three digits long. 2A10 describes two alphanumeric fields, each to be ten characters long.

ANSCII - American National Standard Code for Information Interchange. This is one method of coding alphanumeric characters.

assignment statement - See Equals statement.

auxiliary operation - An offline operation performed by equipment not under control of the central processing unit.

batch processing - The technique of automatically executing a group of programs such that each is completed before the next is started. The DO command stores groups of commands, allowing "unattended" system operation.

benchmark - A problem used to evaluate the performance of hardware or software or both.

bidirectional flow - In flowcharting, flow that can be extended over the same flowline in either direction.

binary program - The form of user's program which is output by the compiler.

bit - A binary digit.

blank - A part of a medium in which no characters are recorded.

bootstrap - A short routine automatically loaded at system startup time (bootstrap switch) to read in system software.

branch - A program stream operation including switching where a selection is made between two or more possible courses of action, depending upon some related fact or condition.

buffer - A temporary storage area usually used for input or output data transfers.

bug - A program error, or a hardware malfunction.

CALL - A program statement that transfers control to a specified subroutine. The subroutine must terminate with a RETURN statement, which returns control to the statement following CALL statement.

Cathode-Ray Tube (CRT) Display - A character television display unit of the operator's console.

central processing unit - A unit of a computer that includes the circuits controlling the interpretation and execution of instructions.

character - A letter, digit, or other symbol used to control or to represent data. See Switch character.

character string - A linear sequence of characters.

clear - Setting an alphanumeric field to space characters, or a decimal field to zeros. In the Data Definition section of a DIBOL program 'C' initially clears a RECORD storage area.

code - Means many things to many programmers. (1) The representation of information, as in ASCII code. (2) A set of instructions or statements called "a piece of code." (3) To code means to write a program.

collating sequence - An ordering assigned to a group of records based on a key item or field within the records. One possible ascending sequence is A-Z, 0-9. Then the descending sequence is 9-0, Z-A.

command - An operator request for Monitor services; usually to be executed immediately.

command string - The characters that make up a complete command.

communication link - The physical means of connecting one location to another for the purpose of transmitting and receiving information.

COMP - The DIBOL compiler program which translates from source programs written in DIBOL language to binary programs which run on the computer.

comments - Notes for people to read, ignored by the compiler. Optional, following a semicolon on any statement line.

connector - A means of representing on a flowchart a break in a line of flow.

data - A representation of information in a manner suitable for communication, interpretation, or processing by either people or machines. In COS 300 systems, data is represented by characters.

data base - The entire set of data files available for processing by COS 300 data management system.

data definition - The specification of record formats in both format programs and source programs. Gives the length of each field, states whether it is alphanumeric or decimal, and may give a field name and initial entry. Data definitions are stored on the systems device, and may be referenced by any other COS 300 program.

data entry - The process of collecting and inputting data into the computer data files. Keyboarding is either key-to-tape or key-to-disk. The systems utility program, BUILD, checks the incoming data for type and length, and writes the records on DECtape or disk. The operator can then print the new data on the line printer to validate the entries.

data independence - When data files can be accessed by any program by referencing a separately stored data definition, data is considered to be independent.

data language - The DIBOL procedural programming language. Source programs written in this language are compiled by COMP, producing binary programs which are executed with the Run-Time system.

data management - The planning, development, and operation of a computer system to mechanize its information flows, and make available the data needed by the user.

debug - To detect, locate, and remove errors or malfunctions from a program or machine.

DEC - Acronym for Digital Equipment Corporation.

decision - A determination of a future action.

decision table - A table of all contingencies that are to be considered in the description of a problem, together with the actions to be taken. Decision tables are sometimes used instead of flowcharts for problem description and documentation.

DECtape reel - Each 4-inch reel contains 260 feet of 3/4-inch wide magnetic tape. Each reel is a logical file of up to 737 blocks of 512 characters each. A large file may consist of up to 63 reels.

detail file - Same as transaction file.

device independence - COS 300 system design permits data files and programs to be stored on either DECtape or disk. At run-time, the operator chooses the most suitable, or available, input and output devices. PIP commands transfer files from one standard device to another.

device names - 3-character abbreviations are used to name the I/O devices.

DT0-DT7	DECtapes 0-7
RK0-RK3	RK5 disk drives
TTY	Terminal printer
KBD	Terminal keyboard
RDR	Paper tape reader
PTP	Paper tape punch
CDR	Card Reader
LPT	Line Printer

DIBOL - Digital's Business Oriented Language is a higher level programming language. It is an integral part of the DEC DATASYSTEM Series 300 Commercial Operating System.

direct access - The process of obtaining data from, or placing data into, a storage device where the time required for each access is independent of the location of the data most recently obtained or places in storage.

directory - See Systems Directory.

disk - A standard mass storage device giving very fast access to data files and programs.

display - A visual presentation of data.

document - A medium and the data recorded on it for human use, for example, a report sheet, a book.

dump - To copy the contents of all or part of storage, usually from core memory to external storage.

EDP - Electronic Data Processing.

END - May be used to terminate DIBOL source programs. Not required by compiler.

end of tape mark - Control characters which mark the physical end of a DECtape reel. When an input file is being read, Monitor detects this EOT mark, and types a message for the operator asking that the next reel in this file be mounted. If an output file, the Monitor asks for another reel.

end of file mark - Identifies the end of the logical file.

equals statement - Manipulates data fields in source programs. Moves data from one field to another, clears fields, calculates the value of arithmetic expressions, and formats data.

error message - An indication that an error has been detected.

field - A specified area in a data record used for either alphanumeric or decimal data, which cannot exceed the specified character length.

file - A collection of records, treated as a logical unit.

FINI - Source language statement required to close output files on disk or DECtape. FINI writes the last record and the end-of-file mark.

fixed-length records - Records in a data file which are all the same length. See also variable-length records.

flowchart - A graphical representation of the definition, analysis, or solution of a problem in which symbols are used to represent operations, data, flow, equipment, etc.

flowchart text - The descriptive information that is associated with flowchart symbols.

flow direction function - The function of linking symbols. The indication of the sequence of available information and executable operations.

flowline - On a flowchart, a line representing a connecting path between flowchart symbols: a line to indicate a transfer of data or control.

format (control) program - A user control program, stored on the systems device, required to run a BUILD, SORT, or UPDATE program. A format program has two parts, Field Descriptor Section and INPUT/OUTPUT Section, which may be stored as two separate programs (or as one) on the systems device.

function - A specific purpose of an entity or its characteristic action.

GO TO - A source program statement that branches to another statement in the Procedure section, usually not to the following statement which would be the normal order of execution.

head of forms - The information printed at the top of a report. May include title, data, page number and column headings.

hit - A successful comparison of two data fields, or keys. See also match.

IF (___.r.___)
GO TO___ - A conditional branch statement in DIBOL source program. If the relationship between two variables is true, the program branches to the label following GOTO. If not true, the next statement is executed.

illegal character - A character that is not valid according to the COS 300 design rules. DIBOL will not accept back slash (\) and back arrow (←) in alphanumeric strings.

inconnector - A connector that indicates a continuation of a broken flowline.

information - The meaning that a human assigns to data by means of the known conventions used in its representation.

information processing - The execution of a systematic sequence of operations performed upon data.

INIT - This statement INITializes a data file. In effect, each INIT opens a file, so that a related XMIT, READ or WRITE can access records from that file.

input - Data flowing into the computer to be processed by a binary program is input data. When the processed data flows out of the computer, it is output data.

Input/Output function - The making available of information for processing (input) or the recording of the processed information (output).

instruction - A program statement that specifies an executable computer operation.

interface - A shared boundary. A hardware component which links two devices, or a storage area accessed by two or more programs. Example: Monitor's Edit Buffer is filled by programs typed in by the operator but taken out of the Edit Buffer and stored on the systems device when a WRITE is given.

I/O - An abbreviation for input/output. (See input/output function.)

item - A group of fields treated as a unit.

job - A set of tasks that makes up a unit of work for a computer. By extension, a job may include all of the necessary data files, systems programs, and instructions that an operator needs to run a job.

jump - A departure from the normal sequence of executing instructions in a computer.

justify - The process of positioning data in a field whose size is larger than the data. In alphanumeric fields, the data is left-justified and any remaining positions are space-filled; in decimal fields the digits are right-justified and any remaining positions are zero-filled.

K - An abbreviation for the prefix kilo. When referring to storage capacity, K=1024 in decimal notation; otherwise, K=1000. COS 300 storage capacities are stated in characters or in record blocks (of up to 510 characters each).

Key - One or more fields within a record used to match or sort a file. If a file is to be arranged by customer name, then the field that contains the customer's names is the key field. In a sort operation, the key fields of two records are compared, and the records are resequenced when necessary.

label - One or more characters (up to a maximum of 6) used to identify a statement in a source program.

leader - The blank section of tape at the beginning of a record.

library - A collection of related files. For example, the collection of inventory control files may form a library, and the libraries used by an organization are known as its data bank.

library routine - A proven routine that is maintained in a program library.

line printer - A high-speed output device that prints all the characters of a line as a unit.

linkage - Coding that connects two separately coded routines.

LN - Monitor command requesting automatic line numbering of a source program or a format program as the program is typed in.

load - To enter data or programs into main core storage.

load-and-go - An operating technique in which there are no stops between the loading and execution phases of a program.

location - Any place where data may be stored.

logical unit - A technique for allocating mass storage facilities at run time. Up to 15 logical units may be assigned at system startup by the SYSGEN program. These areas and their assigned sizes are listed in the Systems Directory. At run time, when Monitor prints "MOUNT filename #1" the operator mounts the file and then types the logical device number.

logical file - A collection of logical records independent of their physical environment. Portions of the same logical record may be located in different physical blocks.

loop - A sequence of instructions that is executed repeatedly until a terminal condition prevails. A commonly used programming technique in processing data records.

magnetic core - The very fast direct-access storage media used as the main internal memory. Contains 2 characters per 12-bit word. It is the equivalent of a two character byte. An 8K core stores up to 16,000 characters.

magnetic tape - A tape with a magnetic surface on which data can be stored by selective polarization of portions of the surface.

main memory - Or main storage. The computer's primary internal storage.

manual input - The entry of data by hand into a device at the time of processing.

mass storage device - A device having large storage capacity, such as DECtapes and disks.

master file - A file that is either relatively permanent, or that is treated as an authority in a particular job.

match - To check for identity between two or more fields.

medium - The material or configuration thereof on which data is recorded: for example, paper tape, cards, magnetic tape.

merge - To combine records from two or more similarly ordered strings into another string that is arranged in the same order. The latter phases of a sort operation.

mnemonic code - To use one or more characters or symbols to depict a well-defined concept. Examples are TTY, RDR and DT4.

Monitor - COS 300 system control program that loads and runs other programs and performs many other useful tasks.

name - The same rules apply to field names, filenames, and statement labels. A name must start with a letter and may use up to 6 significant characters, not including embedded spaces. A name identifies the place where a file, a field, or a statement is stored.

nest - To embed subroutines or loops or data in other subroutines or programs.

NO TRACE - Source language statement. See TRACE.

object program - A compiled program in binary form ready to be loaded and executed.

off line - Equipment or devices that are not under control of the computer.

offline storage - Storage not under control of the central processing unit.

on line - Equipment or services under control of the computer.

online storage - Storage under control of the central processing unit.

operation - The event or specific action performed by a logic element.

outconnector - A connector that indicates a point at which a flowline is broken for a continuation at another point.

output - Data delivered by the computer to external storage.

overlay - The technique of specifying several different record formats for the same data. Special rules apply.

pack - To compress data in a storage medium in such a way that the original data can be recovered. For example, when characters are stored on mass storage media, they are converted to a special 6-bit form, standard 8-bit ANSCII minus 237. Also, fields are packed on magnetic media without separating spaces.

parameter - A variable that is given a constant value for a specific purpose or process.

pass - One cycle of processing a body of data.

patch - To modify a routine or program in a rough or expedient way.

peripheral equipment - Data processing equipment which is distinct from the computer. DECtapes, disks and card readers are examples.

position - In a string, any location that may be occupied by a character.

precision - The degree of discrimination with which a quantity is stated. For example, a three digit numeral discriminates among 1000 possibilities. 6-place numerals are more precise than 4-place numerals. But properly computed 4-place numerals might be more accurate than improperly computed 6-place numerals.

problem definition - A term associated with both the statement and solution phase of a problem and used to denote the transformations of data and the relationship of procedures, data, constraints, environments, etc.

PROC - A data language statement that separates the Data Definition section from the Procedure section. This statement is required in every DIBOL source program. It is a signal to the compiler that the Data Definition section has ended, and that the next data will be the start of the Procedure part of the program. It does not appear in the binary program.

process function - The process of executing a defined operation or group of operations.

processing - A term including any operation or combination of operations on data, where an operation is the execution of a defined action.

program - See source program, binary program, object program, format program.

program library - A Data Center's organized collection of computer programs, off line storage media, and related documentation.

programmer - Person who designs, writes, and tests computer programs.

pseudo-random numbers - A sequence of numbers, computed by an arithmetic process, that is satisfactorily random for a given purpose. Such a sequence may approximate a statistical distribution such as uniform, normal, or gaussian.

punched card - A card punched with a pattern of holes to represent data.

punched tape - A tape on which a pattern of holes or cuts is used to represent data.

random access - Same as direct access.

range - The difference between the highest and lowest values that a quantity or function may assume. For example, the range of decimal numbers that the system can process is -999,999,999,999,999 to +999,999,999,999,999.

READ - A source language statement which inputs records on a direct access device.

read only memory - An equipment option used to store permanently wired instructions.

real time - Use of a computer to guide, control, or acquire data from a related physical process during the actual time that the physical process transpires.

record - A collection of related data fields, and the basic logical unit in data files. A RECORD statement reserves core storage areas for DIBOL data language programs. See also fixed-length and variable-length records. Maximum record size is 510 characters.

record (block) - The basic unit of physical data transfer used primarily to determine storage capacity. A block consists of up to 510 characters.

To determine the physical length in blocks of a data file, the user must add two additional characters for each record in the file, and one additional block of 512 characters for each file (to store the file name). A file must contain an integral number of blocks. Thus if a user is planning to create a data file consisting of 500 records, containing 100 characters each, he must add 500 times 2, plus 512 or a total of 1,512. This file will contain 51,512 characters. To determine the number of blocks this file will occupy, divide by 512: the result is 102.

The length in blocks of programs stored on the systems device is calculated by Monitor and printed in the System Directory in response to DIRECTORY commands. These programs will require 4 additional characters per line to store line numbers.

A DECtape reel contains 737 blocks; an RK08 disk cartridge contains 3240.

RETURN - After CALL, this statement terminates the subroutine and returns control to the statement following CALL.

reverse direction flow - In flowcharting, a flow in a direction other than left to right or top to bottom.

ROM - See read only memory.

security - Protection of data files. Only programs with both the proper filename and data definition can access a file.

segment - To divide a program or file into parts such that the program can be executed without the entire program being in internal storage at any one time.

sequential operation - Performance of operations, such as record processing, one after the other.

serial access - The process of getting data from or putting data into storage where the access time is dependent upon the location of the data most recently obtained or placed in storage. Most magnetic tapes are serially accessed, but DECtapes have fixed addresses, and programs have fast, direct access to their DECtape records.

sign - Positive numbers do not require a sign, but negative numbers are prefixed with the minus sign (-).

significant digit - A digit that is needed for a specific purpose, especially a digit that must be kept to preserve a certain accuracy or precision. Leading zeros are not significant.

simulate - A computer program that represents the behavior of another system. An example would be a program which simulates the behavior of a market when a new product is introduced.

SORT - A utility program which resequences data records within a file into ascending or descending sequence.

source program - A program written in DIBOL data language. These must be translated by the system compiler into DDS 300 machine language before use.

space fill - To fill the remaining character positions in an alphanumeric field with space characters.

special character - A graphic character that is neither a letter, nor a digit, nor a space character.

START - Optional source language statement at beginning of program.

statement - An instruction in a source program.

STOP - A source language statement which terminates a program run, returning control to Monitor.

string - A linear sequence of characters.

stripping - The use of a line across the upper part of a flowchart symbol to signify that a detailed representation of a function is located elsewhere in the same set of flowcharts.

subscripts - A group of data (quantities) arranged in an array. This group or array is referred to by name. Each individual quantity in the array can be referred to in terms of its place by a unique subscript following the array name.

switch character - A single letter specifier in a command. Often follows a slash terminating command.

symbol - A representation of something by reason of relationship or convention.

syntax - The rules governing the structure of a language.

system - An organized collection of software and hardware components, documentation, and methods required to accomplish a specific objective.

system device - A mass storage area reserved for systems programs. This is always logical unit 0.

Systems Directory - A list of systems programs on the systems device with logical device assignments and other useful information.

tape drive - A device that moves tape past a head. Synonymous with tape transport.

terminal - A point in a system at which data can either enter or leave.

TRACE - A source language statement, helpful in debugging, which provides a record or program branches as a program is run. The NO TRACE statement disables the TRACE feature.

transaction file - A file containing relatively transient data to be processed in combination with a master file. For example, in a payroll application, a transaction file indicating hours worked might be processed with a master file containing employee name and rate of pay. Synonymous with detail file.

transmit - To send data from one location and to receive the data at another location.

utility program - A group of systems programs which perform common services, and require format programs. Examples are BUILD, SORT, PIP and PRINT.

variable - A quantity that can assume any of a given set of values.

variable-length record - A file in which the data records are not uniform in length. Specified by V in an INIT statement. Variable length records may be created by DIBOL source programs only, but cannot be processed by utility programs, and direct access to such records by systems programs is impossible.

verify - To determine whether a transcription of data has been accomplished accurately.

word - A string of 12 binary bits, representing two characters.

WRITE - A source language statement which outputs a record to a direct access device.

XMIT - A source language statement which outputs or inputs a record.

zero fill - To fill the remaining character positions in a decimal field with zeros.

HOW TO OBTAIN SOFTWARE INFORMATION

Announcements for new and revised software, as well as programming notes, software problems, and documentation corrections, are published by Software Information Service in the following newsletters.

DIGITAL Software News for the PDP-8 and PDP-12
DIGITAL Software News for the PDP-11
DIGITAL Software News for 18-bit Computers

These newsletters contain information applicable to software available from DIGITAL'S Software Distribution Center. Articles in DIGITAL Software News update the cumulative Software Performance Summary which is included in each basic kit of system software for new computers. To assure that the monthly DIGITAL Software News is sent to the appropriate software contact at your installation, please check with the Software Specialist or Sales Engineer at your nearest DIGITAL office.

Questions or problems concerning DIGITAL'S software should be reported to the Software Specialist. If no Software Specialist is available, please send a Software Performance Report form with details of the problems to:

Digital Equipment Corporation
Software Information Service
Software Engineering and Services
Maynard, Massachusetts 01754

These forms, which are provided in the software kit, should be fully completed and accompanied by terminal output as well as listings or tapes of the user program to facilitate a complete investigation. An answer will be sent to the individual, and appropriate topics of general interest will be printed in the newsletter.

Orders for new and revised software manuals, additional Software Performance Report forms, and software price lists should be directed to the nearest DIGITAL field office or representative. USA customers may order directly from the Software Distribution Center in Maynard. When ordering, include the code number and a brief description of the software requested.

Digital Equipment Computer Users Society (DECUS) maintains a user library and publishes a catalog of programs as well as the DECUSCOPE magazine for its members and non-members who request it. For further information, please write to:

Digital Equipment Corporation
DECUS
Software Engineering and Services
Maynard, Massachusetts 01754

DEC-08-OCSTA-B-D

READER'S COMMENTS

Digital Equipment Corporation maintains a continuous effort to improve the quality and usefulness of its publications. To do this effectively we need user feedback--your critical evaluation of this document.

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

_____

_____

_____

_____

_____

How can this document be improved?

_____

_____

_____

_____

_____

How does this document compare with other technical documents you have read?

_____

_____

_____

_____

_____

Job Title_____Date:_____

Name:_____Organization:_____

Street:_____Department:_____

City:_____State:_____Zip or Country_____

Fold Here

Do Not Tear - Fold Here and Staple

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

Postage will be paid by:

digital

Digital Equipment Corporation
Software Information Service
Software Engineering and Services
Maynard, Massachusetts 01754

DIGITAL EQUIPMENT CORPORATION