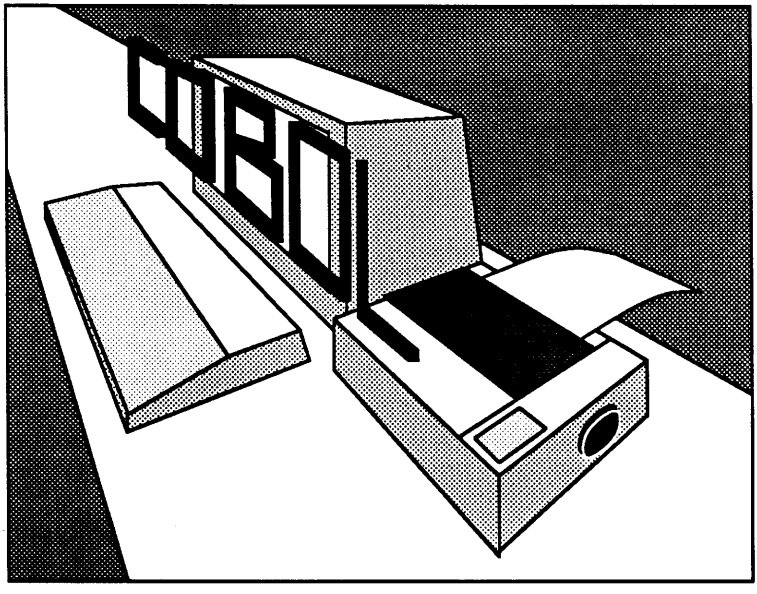


COBOL 128 Software System

By K. A. Alexander

Published by

Abacus Software

Copyright Notice

Warranty

Abacus Software makes no warnings, expressed or implied, as to the fitness of this software package for any particular purpose. In no event will Abacus Software be liable for consequential damages. Abacus Software will replace any copy of this software which is unreadable, if returned within 30 days of purchase. Thereafter, there will be a nominal charge for replacement.

First printing July 1986
Printed in U.S.A.

K.A. Alexander, Visionary Software
Abacus Software, Inc.
P.O. BOX 7219
Grand Rapids, MI. 49510

ISBN# 0-916439-78-X

COBOL 128 Supplement

I COBOL 64 AND COBOL 128 - DIFFERENCES

The manuals for COBOL 64 and COBOL 128 are the same. There are a few minor differences in the two versions:

COBOL 64	COBOL 128
CORD program overlay	in memory
COEDIT program overlay	in memory
COSYN program overlay	in memory
COSYNP program overlay	in memory
load procedure	RUN "VSLOADER"
monitor or television	40 or 80 column monitor/TV

Any references to COBOL 64 throughout this manual also apply to COBOL 128.

II CONVERTING COBOL 64 CODE TO COBOL 128

To convert a COBOL 64 programs to run on the Commodore 128 using COBOL 128, follow the procedures listed below.

1. With COBOL 64 on a C-64 (C-128 in C-64 mode), create a sequential file of the program by using the `CRUNCH` option from the `MAIN MENU`. The filename is prefixed by the letters `CS` (for COBOL Sequential).

2. Load and run COBOL 128. Replace the COBOL 128 distribution diskette with a diskette containing the sequential file.

3. Enter `CRIP` <RETURN> (for CRUNCH input).

4. Now enter the filename without the `CS` prefix followed by <RETURN>.

5. The crunched file is then converted to a format compatible to COBOL 128.

III CONVERTING COBOL 128 CODE TO COBOL 64

To convert a COBOL 128 programs to run on the Commodore 64 using COBOL 64, follow the procedures listed below.

1. With COBOL 128 on a C-128, create a sequential file of the program by using the CRUNCH option from the MAIN MENU. The filename is prefixed by the letters CS (for COBOL Sequential).

2. Load and run COBOL 64. Replace the COBOL 64 distribution diskette with a diskette containing the sequential file.

3. Enter CRIP <RETURN> (for CRUNCH input).

4. Now enter the filename without the CS prefix followed by <RETURN>.

5. The VSEDIT program overlay will now run. When VSEDIT is done, press any key to continue.

6. The crunched file is then converted to a format compatible to COBOL 64.

PREFACE

It is assumed that the readers of this manual are familiar with the Commodore 64 computer and general programming techniques.

The names used in this publication are not of individuals living or otherwise. Any similarity or likeness of the names used in this publication with the names of any individuals, living or otherwise, is purely coincidental and not intentional.

Abacus Software believes that the information described in this manual is accurate and reliable, and much care has been taken in its preparation. However, no responsibility, financial or otherwise, is accepted for any consequences arising out of the use of this material. The information contained herein is subject to change.

ACKNOWLEDGEMENT

COBOL is an industry language and is not the property of any company or group of companies, or of any organization or group of organizations.

No warranty, expressed or implied, is made by any contributor or by the CODASYL Programming Language Committee as to the accuracy and functioning of the programming system and language. Moreover, no responsibility is assumed by any contributor, or by the committee, in connection therewith.

The authors and copyright holders of the copyrighted material used herein,

Flow-matic (trademark of Sperry Rand Corporation). Programming for the UNIVAC I and II, Data Automation Systems copyrighted 1958, 1959, by Sperry Rand Corporation; IBM Commercial Translator Form Number F 28-8013, copyrighted 1959 by IBM;FACT, DSI 27A5260-2760, copyrighted 1960 by Minneapolis-Honeywell,

have specifically authorized the use of this material in whole or in part, in the COBOL specifications. Such authorization extends to the reproduction and use of COBOL specifications in programming manuals or similar publications.

TABLE OF CONTENTS

Chapter 1 - Introduction	1
Chapter 2 - The Nature of COBOL IN GENERAL	2
Chapter 3 - COBOL Advantages	4
Chapter 4 - COBOL 64 Implementation Notes	5
Chapter 5 - Hardware Consideration	6
Chapter 6 - COBOL 64 Source Language	7
Section 1--Program Organization	7
Section 2--Language Concepts	9
Section 3--Editing Format	29
Section 4--IDENTIFICATION DIVISION	32
Section 5--ENVIRONMENT DIVISION	34
Section 6--DATA DIVISION	47
Section 7--PROCEDURE DIVISION	75
ACCEPT	85
ACCEPT-1-KEY	86
ADD	87
CLOSE	89
DEBUG-BREAK	89
DEBUG-TRACE-OFF	89
DEBUG-TRACE-ON	90
DISPLAY	90
DIVIDE	91
EXIT	93
FILTER-NUMERIC	93
GO TO	95
IF	96
MOVE	98
MULTIPLY	102
OPEN	103
PERFORM	105
READ	108
SET	110
STOP RUN	112
SUBTRACT	113
WRITE	115
Chapter 7 -- Start Up/Operating Instructions	119

Chapter 8 -- Main Menu 120
CRUNCH 121
DEBUG 122
START-PROG 123
RESUME-PROG 124
SINGLE-ON 124
SINGLE-OFF 124
EXIT 124
BREAK 1 125
BREAK 2 125
BREAK 3 125
OPTIONS 125
TRACE-ON-LINE 125
TRACE-OFF-LINE 125
TRACE-FAST 126
TRACE-SLOW 126
TRACE-ON 126
TRACE-OFF 127
RESET-OPTIONS 127
EDIT 128
COBOL 64 Statements 128
DIRECTORY 129
LIST 130
DELETE 130
SYNTAX 131
AUTO 132
SAVE 132
RESEQUENCE 133
PRINT-ON 133
PRINT-OFF 133
EXIT 133
GET 134
NEW-NAME 134
NEW-PROG/EDIT 134
PRINT-ON 135
PRINT-OFF 135
RUN 135
SAVE 136

APPENDIX A - Sample Program/Exercises 137
APPENDIX B - Reserved Words 143
APPENDIX C - Language Summary 144
APPENDIX D - Sample Programs 146

CHAPTER 1

INTRODUCTION

This manual provides a complete description of the COBOL 64 System as implemented for use on the Commodore 64 computer or equivalent. The COBOL 64 programming language is designed along the guidelines of the American National Standards Institute (ANSI) X3.23-1974.

The language is an easy to learn subset of the ANSI 1974 standard Level 1 with appropriate extensions to utilize COMMODORE 64 features as well as providing high level program debugging capabilities. With COBOL 64 there is no need for the user to be concerned with machine language, memory addressing or hexadecimal notation. All debugging is performed at the source language (symbolic) level as opposed to the machine language level.

The COBOL 64 software system is a combination of an Editor, Compiler, Interpreter and Symbolic Debugger. These features have been designed with ease of learning and ease of use in mind to provide a powerful programming development tool for general business applications.

Commodore 64 is a registered trademark of Commodore Business Machines, Inc.

CHAPTER 2

THE NATURE OF COBOL IN GENERAL

COBOL is the most widespread commercial programming language in use today. The reasons for its vast success will be discussed below.

The word COBOL is an abbreviation for Common Business Oriented Language. It is a Business Oriented computer language designed for commercial applications. The rules governing the use of the language make it applicable for commercial problems.

COBOL is a computer language that is common to many computers. That is most computers can process a COBOL program with minor variations.

The universality of COBOL, therefore, allows computer users greater flexibility. A company is free to use computers of different manufacturers while retaining a single programming language. Similarly, conversion from one model computer to a more advanced or newer one presents no great problem. Computers of a future generation will also be equipped to use COBOL.

Since its creation in 1959, the COBOL language has undergone extensive refinement in an effort to make it more standardized. The American National Standards Institute (ANSI), an association of computer manufacturers and users, has developed an industry-wide standard COBOL.

Thus the meaning of the word COBOL suggests two of its basic advantages. It is common to most computers, and it is commercially oriented. There are, however, additional reasons why it is such a popular language.

COBOL is an English-like language. All instructions are coded using English words rather than complex codes. To add two numbers together, for example, we use the word ADD. Another example of a COBOL statement is:

```
MULTIPLY HOURS-WORKED BY HOURLY-WAGE GIVING GROSS-WAGES
```

The rules for programming in COBOL conform to many of the rules for writing in English, making it a relatively simple language to learn. It, therefore, becomes significantly easier to train programmers. In addition, COBOL programs are written and tested in far less time than programs written in other computer languages.

Thus the English-like quality of COBOL makes it easy to write programs. Similarly, this quality makes COBOL programs easier to read. Such programs can generally be understood by non-data processing personnel. The business executive who knows little about computers can better understand the nature of a programming job simply by reading a COBOL program.

CHAPTER 3

COBOL ADVANTAGES

The long list of COBOL advantages is derived chiefly from its intrinsic quality of permitting the programmer to state the problem solution in English prose, and thus provide automatic program and system documentation. When users adopt well-chosen data-names before attempting to program a system, maximum documentational advantages of the language described herein are obtained.

To a computer user, COBOL 64 offers the following major advantages:

1. Expeditious means of program implementation providing a high degree of programmer productivity.

2. Accelerated programmer training and simplified retraining requirements.

3. Reduced conversion costs when changing from a computer of one manufacturer to that of another.

4. Significant ease of program modification/enhancements due to the high level of readability.

5. Documentation which facilitates nontechnical management participation in data processing activities.

6. A comprehensive source program diagnostic capability which includes tracing, break points and single step features.

CHAPTER 4

COBOL 64 IMPLEMENTATION NOTES

A program written in COBOL 64, called a source program, is accepted as input by the COBOL 64 software system. The system verifies that each source statement is syntactically correct, and then converts them into an intermediate condensed representation.

The intermediate program can then be executed on the Commodore 64 System using the COBOL 64 Interpreter. The interpreter causes the system hardware to perform the operations specified by the intermediate program and thus the source program.

CHAPTER 5

HARDWARE CONSIDERATIONS

COMMODORE 64 Computer or equivalent (BASIC V2)
One to four Commodore 1541 disk drives or equivalent
One optional Commodore 1525 printer or equivalent

CHAPTER 6

SECTION 1 PROGRAM ORGANIZATION

COBOL 64 SOURCE PROGRAM DIVISIONS

Every COBOL 64 source program must contain these four divisions in the following order:

```
IDENTIFICATION
ENVIRONMENT
DATA
PROCEDURE
```

The IDENTIFICATION DIVISION identifies the program. In addition to required information, the programmer may include such optional pieces of information as the date written and programmer's name for documentation purposes. This division is completely machine-independent.

The ENVIRONMENT DIVISION specifies the equipment being used. It contains computer descriptions and some information about the files the program will use.

The DATA DIVISION contains not only file and record descriptions describing the data files that the program manipulates or creates, but also the individual logical records which comprise these files. The characteristics or properties of the data are described in relation to a standard data format rather than an equipment-oriented format. Therefore, this division is to a large extent, computer-independent. While compatibility among computers cannot be absolutely assured, careful planning in the data layout will permit the same data descriptions, with minor modification, to apply to more than one computer.

The PROCEDURE DIVISION specifies user-supplied steps for computer execution. These steps are expressed in terms of meaningful English words, statements, sentences, and paragraphs. This division of a COBOL 64 program is often referred to as the

"program". In reality, it is only part of the total program, and alone is insufficient to describe the entire program. This is true because repeated references must be made (either explicitly or implicitly) to information appearing in the other divisions. This division, more than any other, allows the user to express thoughts in meaningful English. Concepts of verbs to denote actions, and sentences to describe procedures are basic, as is the use of conditional statements to provide alternative paths of action.

REQUIRED HEADERS

The standard for COBOL 64 requires that a program consist of certain divisions, sections, and fixed paragraph names known as headers.

The following elements are the minimum required for a COBOL 64 program:

```
IDENTIFICATION DIVISION.
 PROGRAM-ID. MINIMUM.
ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
  SOURCE-COMPUTER C64.
  OBJECT-COMPUTER C64.
DATA DIVISION.
PROCEDURE DIVISION.
STARTUP.
   STOP RUN.
```

SECTION 2 LANGUAGE CONCEPTS

GENERAL

As stated in Section 1, COBOL 64 is a language based on English and is composed of words, statements, sentences, and paragraphs. The following paragraphs define the rules to be followed in the creation of this language. The use of the different constructs formed from the created words is covered in subsequent sections of this document.

LANGUAGE DESCRIPTION NOTATION

A nearly universal form of notation exists for COBOL reference manuals. This manual uses that notation as described in the paragraphs that follow.

The apostrophe (') is used to delimit characters with specific meaning. Other than its use in this manual as a delimiter, it has no specific use in the COBOL language.

KEYWORDS

All underlined upper-case words are key words and are required when utilizing related functions. Omissions of key words will cause error conditions at compilation time. An example of key words follows:

```
IF data-name IS [NOT] { NUMERIC    }
                      { ALPHABETIC }
```

The key words are IF, NOT, NUMERIC, and ALPHABETIC.

OPTIONAL WORDS

All upper case words not underlined are optional words included for readability only and may be included or excluded in the source program. In the preceding example, the optional word is IS.

GENERIC TERMS

All lower-case words represent generic terms which are used to represent COBOL words, literals, PICTURE character-strings, comment-entries, or a complete syntactical entry that must be supplied in that format position by the programmer. Where generic terms are repeated in a general format, a number or letter appendage to the term serves to identify that term for explanation or discussion.

Identifier-1 and identifier-2 are generic terms in this example:

```
MOVE identifier-1 TO identifier-2
```

BRACES

The following symbols are braces: { }. When words or phrases are enclosed in braces, a choice of one of the entries must be made. In the previous example in the subsection titled Keywords, either NUMERIC or ALPHABETIC must be included in the statement.

BRACKETS

The following symbols are brackets: [ ]. Words and phrases enclosed in brackets represent optional portions of a statement. A programmer wishing to include the optional feature may do so by including the entry shown between brackets. Otherwise, the optional portion may be omitted. [NOT] in the example titled Keywords, is optional.

LEVEL-NUMBERS

When specific level-numbers appear in data description entry formats those specific level-numbers are required when such entries are used in a COBOL 64 program. In this document, the form `01,02,...,09` is used to indicate level-numbers 1 through 9.

ELLIPSIS

The presence of the ellipsis (three consecutive periods(. . .) within any format indicates the position at which repetition may occur at the programmer's option. The portion of the format that may be repeated is defined in the following paragraph.

The ellipsis applies to the words between the determined pair of delimiters. Given the ellipsis in a clause or statement format, scanning right to left, determine the right bracket or right brace immediately to the left of the ...; continue scanning right to left and determine the logically matching left bracket or left brace.

CHARACTER SET

The COBOL 64 character set for the COBOL 64 System consists of the following 46 characters:

0 through 9
A through Z
blank or space
+ plus sign
- minus sign or hyphen
* asterisk
/ slash
$ currency sign
. period or decimal point
" quotation mark
(left parenthesis
) right parenthesis

CHARACTERS USED FOR WORDS

The character set for words consists of the following 37 characters:

0 through 9
A through Z
- (hyphen)

PUNCTUATION CHARACTERS

The following characters may be used for program punctuation:

" quotation mark
(left parenthesis
) right parenthesis
space or blank
. period

EDITING CHARACTERS

The COBOL 64 System accepts the following characters in editing:

$	currency sign
*	asterisk (check protect)
,	comma
/	slash
B	space or blank insert
0	zero insert
+	plus sign
-	minus sign
CR	credit
DB	debit
Z	zero suppress
.	period

LANGUAGE STRUCTURE

The individual characters of the language are concatenated to form character-strings and separators. A separator may be concatenated with another separator or with a character-string.

A character-string may only be concatenated with a separator. The concatenation of character-strings and separators forms the text of a source program.

SEPARATORS

A separator is a string of one or more punctuation characters. The rules for formation of separators are:

1. The punctuation character space is a separator. Anywhere a space is used as a separator, more than one space may be used.

2. The punctuation character period is a separator when immediately followed by a space.

3. The punctuation character quotation mark is a separator. An opening quotation mark must be immediately preceded by the separator space, a closing quotation mark must be immediately followed by one of the separators space or period followed by a space. Quotation marks may appear only in balanced pairs delimiting nonnumeric literals except when the literal is continued.

4. The punctuation characters right and left parentheses are separators. Parentheses may appear only in balanced pairs of left and right parentheses delimiting subscripts or indices. The right parentheses must be followed by a space.

5. The separator space may optionally immediately follow any separator except the opening quotation mark. In this case, a following space is considered as part of the nonnumeric literal and not as a separator.

Any punctuation character which appears as part of the specification of a PICTURE character-string or numeric literal is not considered as a punctuation character, but rather as a symbol used in the specification of that PICTURE character-string or numeric literal. PICTURE character-strings are delimited only by the separators space or period followed by a space.

The rules established for the formation of separators do not apply to the characters which comprise the contents of nonnumeric literals, comment-entries, or comment lines.

CHARACTER-STRINGS

A character-string is a character of sequence of contiguous characters which forms a COBOL 64 word, literal, PICTURE character-string, or comment-entry. A character-string is delimited by separators.

DEFINITION OF WORDS

A COBOL 64 word is created from a combination of not more than 30 characters, selected from the following:

0 through 9
A through Z
- (hyphen)

A word is ended by a valid separator. A word may not begin or end with a hyphen. (A literal constitutes an exception to these rules, as explained in a paragraph entitled Literals in this section.) A word must begin with an alphabetic character.

A user-defined word is a COBOL 64 word that must be supplied by the user to satisfy the format of a clause or statement.

TYPES OF WORDS

COBOL 64 contains the following word types:

nouns (user-defined words)
verbs
reserved words.

NOUNS

Nouns are divided into special categories:

File-name
Record-name
Data-name
Program-name
Index-name
Paragraph-name

The length of a noun must not exceed 30 characters. For purposes of readability, a noun may contain one or more hyphens. However, the hyphen must neither begin nor end the noun (this does not apply to literals).

All nouns within a given category must be unique, because no other noun in the same source program has identical spelling or punctuation. All user-defined words must begin with an alphabetic character.

File-Name

A file-name is a noun assigned to designate a set of data items. The contents of a file are divided into logical records made up of any consecutive set of data items.

Record-Name

A record-name is a noun assigned to identify a logical record. A record can be subdivided into a set of data items, each distinguishable by a data-name.

Data-Name

A data-name is a noun assigned to identify elements within a record or work area and is used in COBOL 64 to refer to an element of data, or to a defined data area containing data elements.

Index-Name

An index-name is a word that names an index associated with a specific table (refer to Indexing in this section). An index is a register, the contents of which represent the character position of the first character of an element of a table with respect to the beginning of the table.

Paragraph-Name

A paragraph-name is a word which names a paragraph in the PROCEDURE DIVISION.

Verbs

A verb in COBOL 64 is a single word that denotes action, such as ADD, WRITE, or MOVE. All allowable verbs in COBOL 64, with the exception of the word IF, are English verbs. The usage of the COBOL 64 verbs takes place within the PROCEDURE DIVISION.

RESERVED WORDS

A reserved word is a COBOL 64 word that is one of a specified list of words which may be used in COBOL 64 source programs, but must not appear in the programs as user-defined words. Refer to Appendix B, Reserved Words.

These rules apply to the entire COBOL 64 source program; no exceptions exist for specific divisions or statements.

There are two types of reserved words:

Key words
Optional words

KEY WORDS

A key word is a word whose presence is required in a source program. Within each format, such words are upper-case and underlined.

Key words are of three types:

1. Verbs such as ADD and READ.

2. Required words which appear in statement and entry formats.

3. Words which have a specific functional meaning such as SECTION.

OPTIONAL WORDS

Optional words are included in the COBOL 64 language to improve the readability of the statement formats. These optional words may be included or omitted. For example, IF A IS GREATER THAN B...is equivalent to IF A GREATER B...; the inclusion or omission of the words IS and THAN does not influence the logic of the statement.

LITERALS

A literal is an item of data whose value is implied by an ordered set of characters of which the literal is composed. There are two classes of a literal: numeric and nonnumeric.

NUMERIC LITERAL

A numeric literal is a character-string whose characters are selected from the digits 0 through 9, the plus sign (+), the minus sign (-), and/or the decimal point. Numeric literals may be from 1 to 18 digits in length. The rules for the formation of numeric literals are as follows:

1. A numeric literal must contain at least one digit.

2. A numeric literal must not contain more than one sign character. If a sign is used, it must appear as the leftmost character of the literal. If the literal is unsigned, the literal is positive.

3. A numeric literal must not contain more than one decimal point. The decimal point is treated as an assumed decimal point, and may appear anywhere within the literal except as the rightmost character. If the literal contains no decimal point, the literal is an integer. An integer is a numeric literal which contains no decimal point.

   If a literal conforms to the rules of the formation of numeric literals, but is enclosed in quotation marks, it is a nonnumeric literal and is treated as such by the system.

4. The value of a numeric literal is the algebraic quantity represented by the characters in the numeric literal. Every numeric literal belongs to category numeric. Refer to the PICTURE clause in Section 6 for additional information. The size of a numeric literal in standard data format characters is equal to the number

of digits specified by the user. The following are examples of numeric literals:

```
51678
.005
+2.629
-.8479
 6287.92
```

NONNUMERIC LITERAL

A nonnumeric literal may be composed of any allowable character. The beginning and ending of a nonnumeric literal are both denoted by a quotation mark. Any character enclosed within quotation marks is part of the nonnumeric literal. Subsequently, all spaces enclosed within the quotation marks are considered part of the literal. Two consecutive quotation marks within a nonnumeric literal cause a single quotation mark to be inserted into the literal string.

All other punctuation characters are part of the value of the nonnumeric literal rather than separators; all nonnumeric literals belong to category alphanumeric. Refer to the PICTURE clause in Section 6.

A nonnumeric literal cannot exceed 120 characters. Examples of nonnumeric literals are:

Literal on Source Program Level	Literal Stored by System
"THE TOTAL PRICE"	THE TOTAL PRICE
"-2080.479"	-2080.479
"A""B"	A"B

Literals that are used for arithmetic computation must be expressed as numeric literals and must not be enclosed in quotation marks as nonnumeric literals. For example, "4.4" and 4.4 are not equivalent. The system stores the nonnumeric literal as 4.4, whereas the numeric literal would be stored as 0044 if the PICTURE were 999V9,

with the assumed decimal point located between the two fours.

LOGICAL RECORD AND FILE CONCEPTS

The purpose of defining file information is to distinguish between the physical aspects of the file and the conceptual characteristics of the data contained within the file.

PHYSICAL ASPECTS OF A FILE

The physical aspects of a file describe the data as it appears on the input or output media and include such features as:

1. The grouping of logical records within the physical limitations of the file medium.

2. The means by which the file can be identified.

CONCEPTUAL CHARACTERISTICS OF A FILE

The conceptual characteristics of a file are the explicit definition of each logical entity within the file itself. In a COBOL 64 program, the input or output statements refer to one logical record.

It is important to distinguish between a physical record and a logical record. A COBOL 64 logical record is a group of related information, uniquely identifiable, and treated as a unit.

A physical record is a physical unit of information whose size and recording mode are adapted to a particular computer for the storage of data on a input or output device. The size of a physical record is hardware dependent and has no direct relationship to the size of the file of information contained on a device.

The concept of a logical record is not restricted to file data but is carried over into the definition of working storage.

Working storage may be grouped into logical records and defined by a series of record description entries.

RECORD CONCEPTS

The record description consists of a set of data description entries which describe the characteristics of a particular record. Each data description entry consists of a level-number followed by a data-name, if required, followed by a series of independent clauses, as required.

CONCEPT OF LEVELS

A level concept is inherent in the structure of a logical record. This concept arises from the need to specify subdivisions of a record for the purpose of data reference. Once a subdivision has been specified, it may be further subdivided to permit more detailed data referral.

DATA DESCRIPTION CONCEPTS

The most basic subdivisions of a record, those not further subdivided, are called elementary items; consequently, a record is said to consist of a sequence of elementary items, or the record itself may be an elementary item.

In order to refer to a set of elementary items, the elementary items are combined into groups. Each group consists of a named sequence of one or more elementary items. Groups, in turn, may be combined into groups of two or more groups. An elementary item may belong to more than one group.

LEVEL-NUMBERS

A system of level-numbers shows the organization of elementary items and group items. Since records are the most inclusive data items, level-numbers for records start at 01. Less inclusive data items are assigned higher (not necessarily successive) level-numbers not greater in value than 10. There is a special level-number 77 which is an exception to this rule. Separate entries are written in the source program for each level-number used.

A group includes all group and elementary items following it until a level-number less than or equal to the level-number of that group is encountered. All items which are immediately subordinate to a given group item must be described using identical level-numbers greater than the level-number used to describe that group item. Refer to Section 6 under LEVEL-NUMBER for additional information.

CONCEPT OF CLASSES OF DATA

The five categories of data items (refer to the PICTURE clause in Section 6) are grouped into three classes: alphabetic, numeric, and alphanumeric. For alphabetic and numeric, the classes and categories are synonymous. The alphanumeric class includes the categories of alphanumeric edited, numeric edited, and alphanumeric (without editing). Every elementary item, except for an index data item, belongs to one of the classes and also to one of the categories. The class of a group item is treated as alphanumeric regardless of the class of elementary items subordinate to that group item. Table 2-1 shows the relationship of the class and categories of data items.

Table 2-1. Classes of Data

LEVEL OF ITEM	CLASS	CATEGORY
Elementary	Alphabetic	Alphabetic
	Numeric	Numeric
	Alphanumeric	Numeric Edited Alphanumeric Edited Alphanumeric
Nonelementary (Group)	Alphanumeric	Alphabetic Numeric Numeric Edited Alphanumeric Edited Alphanumeric

ALGEBRAIC SIGNS

Algebraic signs fall into two categories: operational signs, which are associated with signed numeric data items and signed numeric literals to indicate algebraic properties; and editing signs, which appear on edited reports to identify the sign of the item.

Operational signs are represented as defined under symbol 'S' of the PICTURE clause. Refer to the PICTURE clause, General Rule 8, the 'S' symbol in Section 6.

Editing signs are inserted into a data item through the use of the sign control symbols of the PICTURE clause.

STANDARD ALIGNMENT RULES

The standard rules for positioning data within an elementary item depend on the category of the receiving item. These rules are:

1. If the receiving data item is described as numeric:

   a. The data is aligned by decimal point and is moved to the receiving character positions with zero fill or truncation on either end as required.

   b. When an assumed decimal point is not explicitly specified, the data item is treated as if it had an assumed decimal point immediately following the rightmost character and is aligned as in step 1a above.

2. If the receiving data item is a numeric edited data item, the data moved to the edited data item is aligned by decimal point with zero fill or truncation at either end as required within the receiving character positions of the data item, except where editing requirements cause replacement of the leading zeros.

3. If the receiving data item is alphanumeric (other than a numeric edited data item), alphanumeric edited or alphabetic, the sending data is moved to the receiving character positions and aligned at the left-most character position in the data item with space fill or truncation to the right, as required.

SUBSCRIPTING

Subscripts can be used only when reference is made to an individual element within a list or table of like elements that have not been assigned individual data-names (refer to the OCCURS clause in Section 6).

The subscript can be represented either by a numeric literal that is an integer or by a data-name. The data-name must be a numeric elementary item that represents an integer. When the subscript is represented by a data-name, the data-name may not be subscripted.

The subscript may be signed and, if signed, must be positive. The lowest possible subscript value is 1. This value points to the first element of the table. The next sequential elements of the table are pointed to by subscripts whose values are 2, 3, and so forth. The highest permissible subscript value, in any particular case, is the maximum number of occurrences of the item as specified in the OCCURS clause.

At the time of execution of a statement which refers to a subscripted table element, each subscript specified is validated. That is, its value must not be less than one or more than the maximum number of occurrences as specified by the corresponding OCCURS clause. If the subscript value is not within this range, an abnormal termination of the program occurs.

The subscript or set of subscripts that identifies the table element is delimited by the balanced pair of separators, left parenthesis and right parenthesis, following the table element data-name. The table element data-name appended with a subscript is called a subscripted data-name or an identifier.

When more than one subscript is required, they are written in the order of successively less inclusive dimensions of the data organization. The maximum number of subscripts is 3.

General Format:

```
data-name ( subscript-1 [ subscript-2 [ subscript-3 ] ] )
```

Example:

In the following record description, to reference the first year, TOTAL-PER-YEAR (1) is written. If data-name YEAR contains the number of the year desired, TOTAL-PER-YEAR (YEAR) is written. If the data item MONTH contains the specific month desired within the year specified by YEAR, TOTAL-PER-MONTH (YEAR MONTH) is written.

```
01  YEAR-TABLE.
    02 TOTAL-PER-YEAR OCCURS 10 TIMES.
      05 TOTAL-PER-MONTH OCCURS 12 TIMES PIC  999.
77  YEAR PIC 99.
77  MONTH PIC 99.
```

INDEXING

References can be made to individual elements within a table of like elements by specifying indexing for that reference. An index is assigned to that level of the table by using the INDEXED BY phrase in the definition of a table. A name given in the INDEXED BY phrase is known as an index-name and is used to refer to the assigned index. The value of an index corresponds to the occurrence number of an element in the associated table. An index-name can be given a value by the execution of a SET statement.

The advantage to indexing is derived by faster execution time when multiple references to the same table element is required. In subscripting a multiply function is needed each time a reference is made. In indexing the multiply only occurs during execution of the SET statement and not each time a reference is made.

An index-name has the same internal representation as an index data item. If a value to be stored in an index-name or in an index data item exceeds the largest value that can be held in that index-name or index data item, the value is truncated according to the rules for the occurrence of a size error condition in an arithmetic statement without a SIZE ERROR phrase.

An index-name assigned to one table may not be used to index another table.

Direct indexing is specified by using an index-name in the form of a subscript. When more than one index-name is required, they are written in the order of successively less inclusive dimensions of the data organization as in subscripting.

At the time of execution of a statement which refers to an indexed table element, the value of each direct index must not be less than a value which corresponds to the beginning of the first occurrence of the table element. Also, the index must not be greater than a value which corresponds to the beginning of the last occurrence of the table element as specified by the corresponding OCCURS clause. If the index value is not within this range, the execution of the program is terminated.

Subscripting is permitted where indexing is permitted.

SECTION 3 EDITING FORMAT

The rules for spacing given in the following description of the reference format take precedence over all other rules for spacing.

FIELD DEFINITIONS

The same format is used for all four divisions of a COBOL 64 program. These divisions must appear in proper order: `IDENTIFICATION`, `ENVIRONMENT`, `DATA`, and `PROCEDURE`. The following paragraphs describe the various fields of this coding form.

SEQUENCE AREA (Record Positions 1-6)

A sequence number, consisting of six digits in the sequence area, must be used for each source program line.

INDICATOR AREA (Record Position 7)

Column 7 has the following functions:

1. If column 7 contains an asterisk (*), the remainder of the record is considered to be a comment and, is not compiled.

2. If column 7 contains a slash (/), the printout is advanced to the top of the next page before printing, and the record is considered to be a comment record. This feature is not available at this time.

3. The presence of a hyphen (-) indicates that the last nonnumeric literal on the previous record is not complete and is continued on this record beginning in Area B (positions 12 through 80).

Nonnumeric literals can be split into two or more records. On the initial record starting from the quotation mark, all information through position 80 is taken as part of the literal, and on the next record a quotation mark must be used to indicate the start of the second part of the literal.

AREA A (Positions 8-11)

DIVISION, SECTION, and PARAGRAPH headers must begin in AREA A. A division header consists of the division name (IDENTIFICATION, ENVIRONMENT, DATA, or PROCEDURE), followed by a space, then the word DIVISION followed by a period.

In the ENVIRONMENT and DATA DIVISIONS, a section header consists of the section-name, followed by a space, and then the word SECTION followed by a period.

A paragraph header consists of the paragraph-name followed by a period. The first sentence of the paragraph may appear on the same line as the paragraph header.

Within the IDENTIFICATION and ENVIRONMENT divisions, the section and paragraph headers are fixed and only the headers shown in this manual are permitted. Within the PROCEDURE DIVISION, the paragraph headers are defined by the user.

Within the DATA DIVISION, the level indicator FD and the level numbers 01 and 77 must each begin in Area A, followed by the associated name and appropriate descriptive information.

AREA B (Positions 12-80)

All entries which are not DIVISION, SECTION, or PARAGRAPH headers; level numbers 01 and 77, or level indicator FD must start in Area B.

When level-numbers are to be indented, each new level-number may begin any number of spaces to the right of Area A.

BLANK LINES

A blank line is one that contains no entries in the Indicator Area, Area A, and Area B. A blank line may appear anywhere in the source program except immediately preceding a continuation line.

PUNCTUATION

The following rules of punctuation apply in COBOL 64 source programs:

1. A sentence must be terminated by a period followed by a space. A period must not appear within a sentence unless it is within a nonnumeric literal or is a decimal point in a numeric literal or PICTURE string.

2. Two or more names in a series must be separated by a space.

3. A space must never be embedded in a name; hyphens are to be used instead. A hyphen must not start or terminate a name. For example:

```
PAY-DAY      (correct)
-PAYDAY      (wrong)
```

SECTION 4 IDENTIFICATION DIVISION

GENERAL

The first division of the COBOL 64 source program is the IDENTIFICATION DIVISION whose function is to identify the source program and the resultant output. In addition, the date the program was written and other pertinent information can be included in the IDENTIFICATION DIVISION.

IDENTIFICATION DIVISION STRUCTURE

The structure of this division follows:

```
 IDENTIFICATION DIVISION.

[PROGRAM-ID.  program-name.

[AUTHOR.  [comment-entry]  ...]

[INSTALLATION.  [comment-entry]  ...]

[DATE-WRITTEN.  [comment-entry]  ...]

[SECURITY.  [comment-entry]  ...]
```

The following rules must be observed in the formation of the IDENTIFICATION DIVISION:

1. The IDENTIFICATION DIVISION must begin with the reserved words IDENTIFICATION DIVISION followed by a period and a space.

2. All paragraph-names must begin in positions 8 through 11 (Area A).

3. The comment-entry must be on the same line as the paragraph name.

4. The comment-entry can consist of any combination of words and literals allowed by the COBOL 64 Editor.

PROGRAM-ID PARAGRAPH

The PROGRAM-ID paragraph gives the name by which a program is identified.

```
PROGRAM-ID.  program-name.
```

The following rules must be observed to form PROGRAM-ID paragraphs.

1. The program-name must conform to the rules for formation of a user-defined word.

2. The PROGRAM-ID paragraph contains the name of the program and must be present in every program.

3. The program-name identifies the source program and all listings pertaining to a particular program.

4. The program-name must be followed by a period and a space.

SECTION 5 ENVIRONMENT DIVISION

GENERAL

The ENVIRONMENT DIVISION is the second division of a COBOL 64 source program. Its function is to specify the computer being used for the program compilation, specify the computer to be used for program execution, and associate files with the computer hardware devices. Furthermore, this division is also used to specify input-output areas to be utilized for each file declared in a program.

ENVIRONMENT DIVISION ORGANIZATION

The ENVIRONMENT DIVISION consists of two sections. The CONFIGURATION SECTION contains the overall specifications of the computer. The INPUT-OUTPUT SECTION deals with files to be used in the object program.

ENVIRONMENT DIVISION STRUCTURE

The structure of this division follows:

```
ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER.  source-computer-entry

OBJECT-COMPUTER.  object-computer-entry

[SPECIAL-NAMES.  special-names-entry]

[INPUT-OUTPUT SECTION.

 FILE-CONTROL.    {file-control-entry }...]
```

The following rules must be observed in the formulation of the ENVIRONMENT DIVISION.

1. The ENVIRONMENT DIVISION must begin with the reserved words ENVIRONMENT DIVISION followed by a period and a space.

2. All entries must begin in Area A (columns 8 through 11).

CONFIGURATION SECTION

The CONFIGURATION SECTION contains information concerning the system to be used for program compilation (SOURCE-COMPUTER), the system to be used for program execution (OBJECT-COMPUTER), and the SPECIAL-NAMES paragraph. The SPECIAL-NAMES paragraph is used to define a special currency sign or decimal point in place of commas.

SOURCE-COMPUTER PARAGRAPH

The SOURCE-COMPUTER paragraph identifies the computer upon which the program is to be compiled.

General Format:

```
SOURCE-COMPUTER.  computer-name.
```

Syntax Rule:

1. The computer-name must be equal to C64 followed by a period and a space.

Example:

```
SOURCE-COMPUTER.  C64.
```

OBJECT-COMPUTER PARAGRAPH

The OBJECT-COMPUTER paragraph identifies the computer on which the program is to be executed.

General Format:

```
OBJECT-COMPUTER.  computer-name.
```

Syntax Rules:

1. Computer-name must be equal to C64 followed by a period and a space.

SPECIAL-NAMES PARAGRAPH

The SPECIAL-NAMES paragraph provides a means of defining a special currency sign or the decimal point in place of commas.

General Format:

```
[CURRENCY SIGN IS literal]

[DECIMAL-POINT IS COMMA] .
```

SPECIAL-NAMES

Syntax Rules:

1. The literal which appears in the CURRENCY SIGN IS literal clause is used in the PICTURE clause to represent the currency symbol. The literal is limited to a single character enclosed in quotation marks and must not be one of the following characters:

   a. Digits 0 through 9.

   b. Alphabetic characters: A B C D L P R S V X Z space

   c. Special characters: * + - , . ; ( ) " / =

   If the CURRENCY SIGN IS clause is not present, the default value dollar sign ($) is used in the PICTURE clause.

   For example: CURRENCY SIGN IS "E"

2. The clause DECIMAL-POINT IS COMMA means that the functions of the comma and period are exchanged in the character-string of the PICTURE clause and in the FILTER-NUMERIC verb. This does not apply to numeric literals.

INPUT-OUTPUT SECTION

The INPUT-OUTPUT section contains information concerning files to be used by the program, and the manner of recording used.

FILE CONCEPTS

In the following paragraphs, concepts of File Types, Organization, Access Mode, I-O Status, INVALID KEY and AT END are discussed pertaining to Sequential and Relative files.

SEQUENTIAL I-O

Sequential I-O provides a capability to access records of a file in established sequence. The sequence is established as a result of writing the records to the file.

Sequential I-O provides full facilities for the FILE-CONTROL and FD entries as specified in the formats of this manual. Within the PROCEDURE DIVISION, Sequential I-O provides full capabilities for the CLOSE, OPEN, READ and WRITE statements.

RELATIVE I-O

Relative I-O provides the capability to access records of a disk file in either a random or sequential manner. Each record in a relative file is uniquely identified by an integer value greater than zero which specifies the record's logical ordinal position in the file.

Relative I-O has full facilities for the FILE-CONTROL and FD entries as specified in the formats of this manual. Within the PROCEDURE DIVISION, the I-O provides full capabilities for the CLOSE, OPEN, READ, and WRITE statements.

ORGANIZATION

Sequential Files are organized such that each record in the file except the first has a unique predecessor record, and each record except the last has a unique successor record. These predecessor-successor relationships are established by the order of WRITE statements when the file is created. Once established, the predecessor-successor relationships do not change except in the case where records are added to the end of the file.

Relative File organization is permitted only on disk storage devices. A Relative File consists of records which are identified by relative record numbers. The file may be thought of as composed of a serial string of areas, each capable of holding a logical record. Each of these areas is denominated by a relative record number. Records are stored and retrieved based on this number. For example, the tenth record is the one addressed by relative record number 10 and is in the tenth record area, whether or not records have been written in the first through the ninth record areas. The maximum size of a relative logical record is 254 characters.

ACCESS MODE

The ACCESS MODE clause specifies the manner in which records are accessed in a file. Sequential and Relative File access methods are discussed in the following paragraphs.

SEQUENTIAL FILES

In the sequential access mode, the sequence in which records are accessed is by the ascending order of ordinal location within the file.

RELATIVE FILE

In the sequential access mode, the sequence in which records are accessed is the ascending order of the relative record numbers of all records which currently exist within the file.

In the random access mode, the sequence in which records are accessed is controlled by the programmer. The desired record is accessed by placing its relative record number in a relative key data item.

I-O STATUS

If the `FILE STATUS` clause is specified in a file control entry, a value is placed into the specified two-character data item during the execution of an `OPEN`, `CLOSE`, `READ`, or `WRITE` statement to indicate to the COBOL 64 program the status of that input-output operation.

The usage of this feature is strongly recommended. The testing of I-O status after each I-O statement will avoid a good deal of confusion when debugging.

STATUS KEY 1

The leftmost character position of the `FILE STATUS` data item is known as status key 1 and is set to a value which indicates one of the following conditions upon completion of the input-output operation.

VALUE	CONDITION
0	Successful Completion
1	At End
2	Invalid Key
3	Permanent Error
9	COBOL 64-Defined Condition:

The above conditions are defined in following text.

SUCCESSFUL COMPLETION

The input-output statement was successfully executed.

AT END

The sequential READ statement is unsuccessfully executed as a result of an attempt to read a record when no next logical record exists in the file.

INVALID KEY

The input-output statement was unsuccessfully executed as a result of one of the following:

1. A READ statement when the contents of the ACTUAL KEY data item are less than 1 or greater than the ordinal number of the last record ever written to the file, or trying to READ a relative file record which was never written to.

2. A WRITE statement when the contents of the ACTUAL KEY data item are less than 1 or greater than the last record allowed to be written because of the specification of a maximum file size.

PERMANENT ERROR

The input-output statement was unsuccessfully executed as the result of a boundary violation for a sequential file or as the result of an input-output error, such as data check parity error.

STATUS KEY 2

The rightmost character position of the FILE STATUS data item is known as status key 2 and is used to further describe the results of the input-output operations. This character contains a value as follows:

1. If no further information is available concerning the input-output operation, then status key 2 contains a value of 0.

2. When status key 1 contains a value of 2 indicating an INVALID KEY condition, status key 2 is used to designate the case of that condition as follows:

   a. A value of 3 in status key 2 indicates no record found. An attempt is made to access a record, identified by a key, but that record does not exist in the file.

   b. A value of 4 in status key 2 indicates a boundary violation. An attempt was made to write beyond the externally defined boundaries of the file.

3. When status key 1 contains a value of 9 indicating a COBOL 64-defined condition, the value of status key 2 indicates the condition as follows:

STATUS KEY 2

VALUE CONDITION

0 This is a Commodore exception which is displayed on the screen. See your Commodore User's Guide for additional information.

1 Attempted to OPEN or CLOSE when file was already opened or closed.

2 Device not present or powered on.

3 Attempted READ or WRITE when file was not opened or not opened properly. A READ must be opened INPUT or I-O. A WRITE must be opened OUTPUT or I-O.

4 Attempted READ when previous READ resulted in an end of file condition.

VALID COMBINATIONS OF STATUS KEYS 1 and 2

The valid permissible combinations of the values of status key 1 and status key 2 are shown in Table 5-1.

TABLE 5-1. STATUS KEY COMBINATIONS

STATUS KEY 1	STATUS KEY 2	
0	0	Successful completion
1	0	AT END
2	3	INVALID KEY, no record found
2	4	INVALID KEY, boundary violation
3	0	Permanent error
9	0	Commodore error
9	1	OPEN CLOSE error
9	2	Devise not present/ready
9	3	READ or WRITE with OPEN error
9	4	READ after end of file

FILE-CONTROL PARAGRAPH

The FILE-CONTROL paragraph names each file and allows specification of other file-related information.

General Format:

```
FILE-CONTROL.    {file-control-entry }  ...
```

FILE CONTROL ENTRY

The file control entry names a file and may specify other file-related information.

```
INPUT-OUTPUT SECTION.
FILE-CONTROL.
          SELECT  filename
                                  ┌ DISK-1541   ┌ DRIVE-8  ┐
                ASSIGN  TO        │             │ DRIVE-9  │
                                  │             │ DRIVE-10 │
                                  │             └ DRIVE-11 ┘
                                  └ PRINTER-1525
[ [ ORGANIZATION IS SEQUENTIAL ]
[ [ ACCESS MODE IS SEQUENTIAL  ] ]

[ ORGANIZATION IS RELATIVE
  ACCESS MODE IS { SEQUENTIAL [RELATIVE KEY IS data-name-2] }
                 { RANDOM     RELATIVE KEY IS data-name-2   }
]
[ FILE STATUS IS data-name-1 ].
```

Syntax Rules:

1. The SELECT clause must be specified first in the file control entry. The clauses which follow the SELECT clause may appear in any order.

2. Each file described in the DATA DIVISION must be named only once with a file-name in the FILE-CONTROL paragraph. Each file specified in the file control entry must have a file description entry in the DATA DIVISION.

3. If the ACCESS MODE clause is not specified, the ACCESS MODE IS SEQUENTIAL clause is implied.

4. Data-name-1 must be defined in the DATA DIVISION as a two-character data item of the category alphanumeric and must not be defined in the FILE SECTION or the COMMUNICATION SECTION.

5. If no ORGANIZATION IS clause is specified, the ORGANIZATION IS SEQUENTIAL clause is implied.

6. The RELATIVE KEY phrase may be specified only for disk storage files.

7. Data-name-2 must not be defined in a record description entry associated with that file-name.

8. The data item referenced by data-name-2 must be defined as an unsigned integer.

General Rules:

1. The ASSIGN clause specifies the association of the file referenced by file-name to a storage medium. For Relative and Indexed Files, the storage medium must be DISK-1541.

2. The ORGANIZATION clause specifies the logical structure of a file. The file organization is established at the time a file is created and cannot subsequently be changed.

3. When the access mode of a Relative File is sequential, records in the file are accessed in the order of ascending relative record numbers of existing records in the file.

4. When the FILE STATUS clause is specified, a value is moved by the COBOL 64 system into the data item specified by data-name-1 after the execution of every statement that references that file either explicitly or implicitly. This value indicates the status of execution of the statement. Refer to I-O Status in this section for additional information.

5. If the access mode of a Relative File is random, the value of the RELATIVE KEY data item indicates the record to be accessed.

6. All records stored in a Relative File are uniquely identified by relative record numbers. The relative record number of a given record specifies the record's logical ordinal position in the file. The first logical record has a relative record number of 1, and subsequent logical records have relative record numbers of 2, 3, 4, and so forth.

7. In a Relative File, the data item specified by data-name-2 is used to communicate a relative record number between the program and the COBOL 64 system.

SECTION 6 DATA DIVISION

GENERAL

The DATA DIVISION describes the data that the object program is to accept as input, to manipulate, to create, or to produce as output. Data to be processed belongs to these three categories:

1. That which is contained in files and enters or leaves the internal memory of the computer from a specified area or areas.

2. That which is developed internally and placed into intermediate or working storage, or placed into specific format for output reporting purposes.

3. Constants which are defined by the user.

DATA DIVISION ORGANIZATION

The DATA DIVISION, which is one of the required divisions in a program, is subdivided into sections. These are FILE and WORKING-STORAGE.

The FILE SECTION defines the structure of data files. Each file is defined by a file description entry and one or more record descriptions. Record descriptions are written immediately following the file description entry.

The WORKING-STORAGE SECTION describes records and noncontiguous data items which are not part of external data files but are developed and processed internally. It also describes data items whose values are preassigned in the source program.

DATA DIVISION STRUCTURE

The following structure shows the general format of the sections of the DATA DIVISION, and defines the order of presentation in the source program.

```
DATA DIVISION.
[
FILE SECTION.

[file-description-entry  [record-description-entry]  ...]  ...]
[
WORKING-STORAGE SECTION.

[
 77-level-description-entry
 record-description-entry ]...]
```

FILE SECTION

In a COBOL 64 program, the file description entry (FD) represents the highest level of organization in the FILE SECTION. The FILE SECTION header is followed by a file description entry consisting of a level indicator (FD), a file-name, and a series of independent clauses. The FD clauses specify the size of the logical records, the presence or absence of label records and the value of label items. The entry is terminated by a period.

RECORD DESCRIPTION

A record description consists of a set of data description entries which describe the characteristics of a particular record. Each data description entry consists of a level-number followed by a data-name (if required), followed by a series of independent clauses as required.

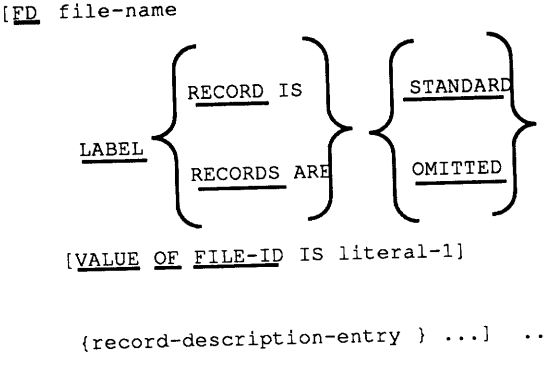
Examples:

```
01  DATA-ITEM-ONE          PICTURE IS X(10).
03  LINE-COUNTER           PIC  999.
```

A record description has a hierarchical structure and, therefore, the clauses used with an entry may vary considerably, depending upon whether or not it is followed by subordinate entries. The structure of a record description is defined in Concepts of Levels, Section 2, while the elements allowed in a record description are shown in the data description structure.

FILE DESCRIPTION STRUCTURE

The file description entry furnishes information concerning the physical structure and identification pertaining to a given file.



LABEL RECORDS

The LABEL RECORDS clause specifies whether labels are present.

Syntax Rules:

This clause is required. The clause is treated as documentation only.

VALUE OF

The `VALUE OF` clause specifies the file identification of a disk file.

Syntax Rules:

Literal-1 must be a nonnumeric literal not greater than 19 characters. The first three characters must be `@0:` if the "replace" feature is desired. Note that relative files cannot be "replaced". See your Commodore user's guide for more detailed information.

General Rules:

1. For an input file, the appropriate label routine checks to see if the disk drive contains a file name equal to the value of literal-1.

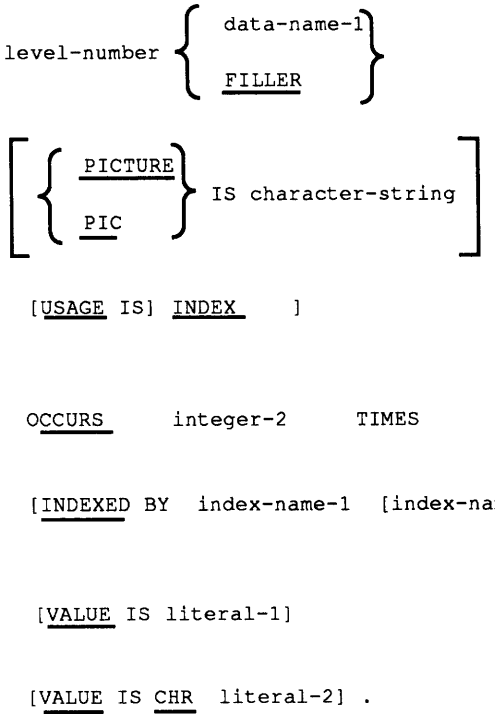
   For an output file, at the appropriate time, the value of literal-1 is used to create the disk file name.

Example: `VALUE OF FILE-ID IS "MY-FILE"`

DATA DESCRIPTION STRUCTURE

A data description entry specifies the characteristics of a particular item of data.

General Format:



Syntax Rules:

1. The level-number may be any number from 01 through 10 or 77.

2. The clauses may be written in any order with one exception: the data-name-1 or FILLER clause must immediately follow the level-number.

3. The PICTURE clause must be specified for every elementary item except an index data item, in which case, use of this clause is prohibited. The PICTURE clause and character-string must be on the same line.

General Rules:

1. The PICTURE clauses must not be specified except for an elementary data item.

DATA-NAME OR FILLER

A data-name specifies the name of the data being described. The word FILLER specifies an elementary item of the logical record that cannot be referred to explicitly.

General Format:

```
{ data-name }
{ FILLER    }
```

Syntax Rules:

1. In the FILE and WORKING-STORAGE SECTIONS, a data-name or the key word FILLER must be the first word following the level-number in each data description entry.

General Rules:

1. The key word FILLER may be used to name an elementary item in a record. Under no circumstances can a FILLER item be referred to explicitly.

2. The key word FILLER is not allowed with a level 77 item or with a VALUE clause.

LEVEL-NUMBER

The level-number shows the hierarchy of data within a logical record. In addition, it is used to identify entries for working storage items.

General Format:

```
level-number
```

Syntax Rules:

1. A level-number is required as the first element in each data description entry.

2. Data description entries subordinate to an FD must have level-numbers with the values 01 through 10. Refer to the FD file description in the paragraph entitled File Description Structure.

3. Data description entries in the WORKING-STORAGE SECTION must have level-numbers with the values 01 through 10.

General Rules:

1. The level-number 01 identifies the first entry in each record description. Less inclusive groupings are given higher numbers (not necessarily successive) up to a limit of 10.

2. A special level-number has been assigned to entries where there is no real concept of level: the level-number 77 is assigned to identify noncontiguous working storage data items.

3. Multiple level 01 entries subordinate to any given level indicator (FD) represent implicit redefinitions of the same area.

Examples:

The following is an example of record layout which corresponds to Figure 6-3 showing a record description and the use of level numbers.

```
                    STUDENT RECORD

STUDENT NO.          NAME              GRADE BIRTH DATE
                  LAST      FIRST            mth day yr


 01   STUDENT-REC.
      03  STUDENT-NO                  PIC 9(6).
      03  STUDENT-NAME.
          05  LAST-NAME               PIC X(8).
          05  FIRST-NAME              PIC X(5).
      03  GRADE                       PIC 99.
      03  BIRTH-DATE.
          05  BIRTH-MONTH             PIC 99.
          05  BIRTH-DAY               PIC 99.
          05  BIRTH-YEAR              PIC 99.
```

Figure 6-3 Level Numbers

OCCURS

The OCCURS clause eliminates the need for separate entries for repeated data items and supplies information required for the application of subscripts or indices.

General Format:

```
OCCURS integer-2 TIMES

  [INDEXED BY index-name-1 [  index-name-2] ...]
```

Syntax Rules:

1. An INDEXED BY phrase is required if the subject of this entry, or an entry subordinate to this entry, is to be referred to by indexing.

2. The OCCURS clause cannot be specified in a data description entry that has an 01 or 77 level-number.

3. Index-name-1, index-name-2,... must be unique words within the program.

4. Integer-2 cannot be zero and cannot be greater than 9,999.

General Rules:

1. The OCCURS clause is used in defining tables and other homogeneous sets of repeated data items. Whenever the OCCURS clause is used, the data-name which is the subject of this entry must be either subscripted or indexed whenever it is referred to in a statement. Further, if the data-name associated with the OCCURS clause is the name of a group item, then all data-names belonging to the group must be subscripted or indexed when used as operands. Refer to Subscripting, Indexing, and Identifier in Section 2.

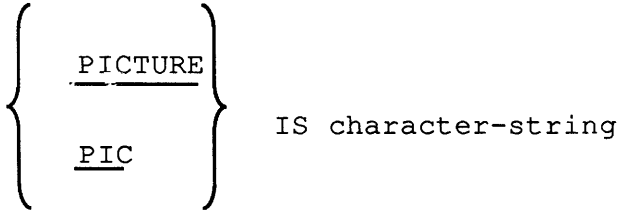
2. Except for the OCCURS clause, all data description clauses associated with an item whose description includes an OCCURS clause apply to each occurrence of the item described.

3. The VALUE clause is not allowed with the OCCURS clause.

PICTURE

The PICTURE clause describes the general characteristics and editing requirements of an elementary item.

General Format:



Syntax Rules:

1. A PICTURE clause can be specified only at the elementary item level.

2. A character-string consists of certain allowable combinations of characters in the COBOL 64 character set used as symbols. The allowable combinations determine the category of the elementary item.

3. The maximum number of characters allowed in the character-string is 30.

4. The PICTURE clause must appear in every elementary item except those items whose USAGE is declared as INDEX.

5. PIC is an abbreviation for PICTURE.

General Rules:

1. There are five categories of data that can be described with a PICTURE clause; alphabetic, numeric, alphanumeric, alphanumeric edited, and numeric edited.

2. To define an item as alphabetic:

   a. The PICTURE character-string can only contain the symbol 'A'.

   b. The item contents, when represented in standard data format, must be any combination of the 26 letters of the English alphabet and the space from the computer character set.

3. To define an item as numeric:

   a. The PICTURE character-string can only contain the symbols '9', 'S', and 'V'. The number of digit positions that can be described by the PICTURE character-string must range from 1 to 18 inclusive.

   b. If unsigned, the item contents must be a combination of the numerals '0', '1', '2', '3', '4', '5', '6', '7', '8', and '9'; if signed, the item may also contain a '+' or '-'.

4. To define an item as alphanumeric:

   a. The PICTURE character-string is restricted to certain combinations of the symbols 'A', 'X', '9', and the item is treated as if the character-string contained all X's. A PICTURE character-string which contains all A's or all 9's does not define an alphanumeric item.

b. The item contents are allowable characters in the computers.

5. To define an item as alphanumeric edited:

a. The PICTURE character-string is restricted to certain combinations of the following symbols: 'A', 'X', '9', 'B', '0', and '/'.

1) The character-string must contain at least one 'B' and at least one 'X' or at least one '0' (zero) and at least one 'X' or at least one '/' (stroke) and at least one 'X'.

2) The character-string must contain at least one '0' (zero) and at least one 'A' or at least one '/' (stroke) and at least one 'A'.

b. The contents are allowable characters in the computer character set.

6. To define an item as numeric edited:

a. The PICTURE character-string is restricted to certain combinations of the symbols 'B', '/', 'V', 'Z', '0', '9', ',', '.', '*', '+', '-', 'CR', 'DB', and the currency symbol ($). The allowable combinations are determined from the order of precedence of symbols and the editing rules.

1) The number of digit positions that can be represented in the PICTURE character-string must range from 1 to 18 inclusive.

2) The character-string must contain at least one '0', 'B', '/', 'Z', '*', '+', ', ', '. ', '-', 'CR', 'DB', or currency symbol.

b. The contents of the character positions of these symbols that are allowed to represent a digit in standard data format must be one of the numerals.

7. The size of an elementary item, where size means the number of character positions occupied by the elementary item in standard data format, is determined by the number of allowable symbols that represent character positions. An integer which is enclosed in parentheses following the symbols: 'A', ', ', 'X', '9', 'Z', '*', 'B', '/', '0', '+', '-', or '$' indicates the number of consecutive occurrences of the symbol. The maximum value of this integer is 9,999. The following symbols may appear only once in a given PICTURE: 'S', 'V', '. ', 'CR', and 'DB'.

8. The functions of the symbols used to describe an elementary item are explained as follows:

'A'

Each letter 'A' in the character-string represents a character position which can contain only a letter of the alphabet or a space.

'B'

Each letter 'B' in the character-string represents a character position into which the space character will be inserted.

'S'

The letter 'S' is used in a character-string to indicate the presence of an operational sign in the internal representation of a numeric data item. It must be the first (leftmost) character in the character-string.

When an operational sign is specified the sign is maintained and expected in the zone of the most significant (leftmost) character. When the data item is the receiving field in an arithmetic statement the four zone bits are set to binary 0101 for negative values and to binary 0100 for positive values. When the data item

is used in an algebraic comparison or operation to supply an algebraic value, only the most significant zone being a binary 0101 will cause the value of the data item to be considered negative. Only the zone values 0100 and 0101 will qualify the data item as being NUMERIC if tested by the NUMERIC class condition.

'V'

The letter 'V' is used in a character-string to indicate the location of the assumed decimal point and may only appear once in a character-string. The 'V' does not represent a character position and is not counted in the size of the elementary item. When the assumed decimal point is to the right of the rightmost symbol in the string, the 'V' is redundant.

'X'

Each letter 'X' in the character-string is used to represent a character position which contains any allowable character in the character set.

'Z'

Each letter 'Z' in a character-string may only be used to represent the leftmost leading numeric character positions which are replaced by a space character when the contents of that character positions are zero. Each 'Z' is counted in the size of the item.

'9'

Each numeral '9' in the character-string represents a character position which contains a numeral and is counted in the size of the item.

'0'

Each numeral '0' in the character-string represents a character position into which the numeral zero is inserted. The '0' is counted in the item.

'/'

Each stroke '/' in the character-string represents a character position into which the stroke character is inserted. The '/' is counted in the size of the item.

','

Each comma ',' in the character-string represents a character position into which the character ',' is inserted. This character position is counted in the size of the item. The insertion character ',' must not be the last character in the PICTURE character-string.

'.'

When the character period '.' appears in the character-string it is an editing symbol which represents the decimal point for alignment purposes and in addition, represents a character position into which the character '.' is inserted. The character '.' is counted in the size of the item. For a given program the functions of the period and comma are exchanged if the clause DECIMAL-POINT IS COMMA is stated in the SPECIAL-NAMES paragraph. In this exchange the rules for the period apply to the comma and the rules for the comma apply to the period when appearing in a PICTURE clause. The insertion character '.' must not be the last character in the PICTURE character-string.

'+' '-' 'CR' 'DB'

These symbols are used as editing sign control symbols and when used, represent the character position into which the editing sign control symbol will be placed. The symbols are mutually exclusive in any one character-string and each character used in the symbol is counted in determining the size of the data item.

'*'

Each asterisk '*' in the character-string represents a leading numeric character position into which an asterisk is placed when the contents of that position are zero. Each '*' is counted in the size of the item.

'CS'

The currency symbol '$' in the character-string represents a character position into which a currency symbol is placed. The currency symbol in a character-string is represented by either the dollar sign '$' or by the character specified in the CURRENCY SIGN clause in the SPECIAL-NAMES paragraph. The currency symbol is counted in the size of the item.

EDITING RULES:

1. There are two general methods of performing editing in the PICTURE clause, either by insertion or by suppression and replacement. The four types of insertion editing available are:

   a. Simple insertion
   b. Special insertion
   c. Fixed insertion
   d. Floating insertion

   There are two types of suppression and replacement editing:

   a. Zero suppression and replacement with spaces
   b. Zero suppression and replacement with asterisks

2. The type of editing which may be performed upon an item is dependent upon the category to which the item belongs. Table 6-1 specifies which type of editing may be performed upon a given category:

Table 6-1. Editing for Each Item Category

Category	Type of Editing
Alphabetic	None
Numeric	None
Alphanumeric	None
Alphanumeric Edited	Simple insertion '0', 'B', and '/'
Numeric Edited	All, subject to Editing Rule 3

3. Floating insertion, and editing by zero suppression and replacement, are mutually exclusive in a PICTURE clause. Only one type of replacement may be used with zero suppression in a PICTURE clause.

4. Simple Insertion Editing. The ',' (comma), 'B' (space), '0' (zero), and '/' (stroke) are used as the insertion characters. The insertion characters are counted in the size of the item and represent the position in the item into which the character is inserted.

5. Special Insertion Editing. The '.' (period) is used as the insertion character. In addition to being an insertion character it also represents the decimal point for alignment purposes. The insertion character, used for the actual decimal point, is counted in the size of the item. The use of the assumed decimal point, represented by the symbol 'V' and the actual decimal point, represented by the insertion character, in the same PICTURE character-string is disallowed. The result of special insertion editing is the appearance of the insertion character in the item in the same position as shown in the character-string.

6. Fixed Insertion Editing. The currency symbol and the editing sign control symbols, '+', '-', 'CR', 'DB', are the insertion characters. Only one currency symbol and only one of the editing sign control symbols can be used in a given PICTURE character-string. The symbols 'CR' or 'DB' always represent two character positions in determining the

size of the item and must represent the rightmost character positions counted in the size of the item. The symbol '+' or '-' when used, must be either the leftmost or rightmost character position to be counted in the size of the item. The currency symbol must be the leftmost character position to be counted in the size of the item except that it can be preceded by either a '+' or '-' symbol. Fixed insertion editing results in the insertion character occupying the same character position in the edited item as in the PICTURE character-string. Editing sign control symbols produce the results shown in Table 6-2 depending upon the value of the data item.

Table 6-2. Editing of Sign Control Symbols

Editing Symbol in PICTURE Character-String	Result Data Item Positive or Zero	Data Item Negative
+	+	-
-	space	-
CR	2 spaces	CR
DB	2 spaces	DB

7. Floating Insertion Editing. The currency symbol and editing sign control symbols '+' or '-' are the floating insertion characters and are mutually exclusive in a given PICTURE character-string.

   Floating insertion editing is indicated in a PICTURE character-string by using a string of at least two of the floating insertion characters. This string of floating insertion characters may contain any of the fixed insertion symbols or have fixed insertion characters immediately to the right. These simple insertion characters are part of the floating string.

   The leftmost character of the floating insertion string represents the leftmost limit of the floating symbol in the data item. The rightmost character of the floating

string represents the rightmost limit of the floating symbols in the data item.

The second floating character from the left represents the leftmost limit of the numeric data that can be stored in the data item. Nonzero numeric data may replace all the characters at or to the right of this limit.

In a PICTURE character-string, there are only two ways of representing floating insertion editing. One is to represent any or all of the leading numeric character positions on the left of the decimal point by the insertion character. The other is to represent all of the numeric character positions in the PICTURE character-string by the insertion character.

If the insertion characters are only to the left of the decimal point in the PICTURE character-string, the result is that a single floating insertion character is placed into the character position immediately preceding either the decimal point or the first nonzero digit in the data represented by the insertion symbol string, whichever is farther to the left in the PICTURE character-string. The character positions preceding the insertion character are replaced with spaces.

If all numeric character positions in the PICTURE character-string are represented by the insertion character, the result depends upon the value of the data. If the value is zero, the entire data item will contain spaces. If the value is not zero, the result is the same as when the insertion character is only to the left of the decimal point.

To avoid truncation, the minimum size of the PICTURE character-string for the receiving data item must be the number of characters in the sending data item, plus the number of nonfloating insertion characters being edited into the receiving data item, plus one for the floating insertion character.

8. Zero Suppression Editing. The suppression of leading zeroes in numeric character positions is indicated by the use of the alphabetic character 'Z' or the character '*' (asterisk) as suppression symbols in a PICTURE character-string. These symbols are mutually exclusive in a given PICTURE character-string. Each suppression symbol is counted in determining the size of the item. If 'Z' is used the replacement character is the space, and if the asterisk is used the replacement character will be '*'.

   Zero suppression and replacement is indicated in a PICTURE character-string by using a string of one or more of the allowable symbols to represent leading numeric character positions which are to be replaced when the associated character position in the data contains a zero. Any of the simple insertion characters embedded in the string of symbols or to the immediate right of this string are part of the string.

   In a PICTURE character-string, there are only two ways of representing zero suppression. One is to represent any or all of the leading numeric character positions to the left of the decimal point by suppression symbols. The other is to represent all of the numeric character positions in the PICTURE character-string by suppression symbols.

   If the suppression symbols appear only to the left of the decimal point, any leading zero in the data which corresponds to a symbol in the string is replaced by the replacement character. Suppression terminates at the first nonzero digit in the data represented by the suppression symbol string or at the decimal point, whichever is encountered first.

   If all numeric character positions in the PICTURE character-string are represented by suppression symbols, and the value of the data is not zero, the result is the same as if the suppression characters were only to the left of the decimal point. If the value is zero and the suppression symbol is 'Z', the entire

data item will be spaces. If the value is zero and the suppression symbol '*', the data item will be all asterisks except for the actual decimal point.

9. The symbols '+', '-', '*', 'Z', and '$', when used as floating replacement characters, are mutually exclusive within a given character-string.

10. At least one of the symbols 'A', 'X', 'Z', '9' or '*', or at least two of the symbols '+', '-' or 'cs' must be present in a PICTURE string.

Examples:

The following Table 6-3 demonstrates the editing function of the PICTURE Clause.

Table 6-3. Editing Application of the PICTURE Clause

Source Area			Receiving Area	
Picture		Data	Editing Picture	Edited Data
9(5)		12345	00999.00	00345.00
9(3)V99		12345	999.BB	123.
S9(5)	(+)	12345	$$$$$$.99CR	$12345.00
S99V9(3)	(-)	12345	------.99	-12.34
S9(5)	(-)	12345	$$$$$$.99CR	$12345.00CR
S9(5)V	(-)	12345	-ZZZZ9.99	-12345.00
9(5)		12345	BBB99.99	45.00
S9(5)	(+)	12345	ZZZZ9.99-	12345.00
S9(5)	(-)	12345	ZZZZ9.99-	12345.00-
9(3)V99		12345	999.00	123.00
S9(5)	(-)	00123	--99999.99	-00123.00
S9(5)	(+)	12345	ZZZZ9.99+	12345.00+
9(3)V99		00001	$$$,$$$.$$	$.01
9(5)		00000	$ZZ,ZZZ.ZZ	
9(5)		12345	$$$,$$9.99	$12,345.00
9(3)V99		00001	$ZZ,ZZZ.99	$ .01
9(3)V99		12345	$$$,$$9.99	$123.45
9(3)V99		00012	$ZZ,ZZ9.99	$ 0.12
9(5)		00123	$**,**9.99	$***123.00
9(5)		00000	$**,***.**	*******.**
9(5)		01234	$**,**9.99	$*1,234.00
9(5)		00000	$$$,$$$.$$	
9(3)V99		12345	$ZZ,ZZ9.99	$ 123.45
9(5)		00000	$$$,$$9.99	$0.00
V9(5)		12345	$ZZ,ZZ9.99	$ 0.12
V9(5)		12345	$$$,$$9.99	$0.12
9(5)		12345	$ZZ,ZZ9.99	$12,345.00

USAGE

The USAGE clause specifies the format of a data item in the computer storage.

General Format:

```
[USAGE IS] INDEX
```

Syntax Rules:

1. An index data item can be referenced explicitly only in a SET statement or a relation condition.

2. The OCCURS, PICTURE, or VALUE clauses cannot be used to describe group or elementary items described with the USAGE IS INDEX clause.

General Rules:

1. An elementary item described with the USAGE IS INDEX clause is called an index data item and contains a value which must correspond to an occurrence number of a table element.

   The group item is considered to be a group data item whose class is alphanumeric and may be referenced any place in the syntax acceptable for such an item. The size of the group item is considered to be in terms of characters, six characters for each subordinate index data item.

2. An index data item can be part of a group which is referred to in an MOVE, or input-output statement, in which case no conversion takes place.

VALUE

The VALUE clause defines the value of constants and the initial value of working-storage items.

General Format:

Format 1:

```
VALUE IS literal
```

Format 2:

```
VALUE IS CHR literal-2
```

Syntax Rules:

1. The VALUE clause cannot be stated for any item that has the key word FILLER or one which has an OCCURS clause or subordinate to an item which has an OCCURS clause.

2. A signed numeric literal must have an associated signed numeric PICTURE character-string.

3. All numeric literals in a VALUE clause of an item must have a value which is within the range of values indicated by the PICTURE clause, and must not have a value which would require truncation of nonzero digits. Nonnumeric literals in a VALUE clause of an item must not exceed the size indicated by the PICTURE clause.

4. Format 2 VALUE clauses must contain a PICTURE clause with a character-string equal to one X. Literal-2 must be an unsigned integer between 0 and 255.

General Rules:

1. The VALUE clause must not conflict with other clauses in the data description of the item or in the data description within the hierarchy of the item. The following rules apply:

   a. If the category of the item is numeric, the literal in the VALUE clause must be numeric. If the literal defines the value of a working-storage item, the literal is aligned in the data item according to the standard alignment rules. Refer to Standard Alignment Rules in Section 2.

   b. If the category of the item is alphabetic, alphanumeric, alphanumeric edited or numeric edited, all literals in the VALUE clause must be nonnumeric literals. The literal is aligned in the data item as if the data item had been described as alphanumeric. (Refer to Standard Alignment Rules in Section 2.) Editing characters in the PICTURE clause are included in determining the size of the data item (refer to the PICTURE clause in this section) but have no effect on initialization of the data item. Therefore, the VALUE for an edited item is presented in an edited form.

2. Rules governing the use of the VALUE clause differ with the respective sections of the DATA DIVISION:

   a. In the FILE SECTION, the VALUE clause may not be used.

   b. In the WORKING-STORAGE SECTION the VALUE clause is used to specify the initial value of the data item, in which case the clause causes the item to assume the specified value at the start of the object program. If the VALUE clause is not used in an item description, the initial value is undefined.

3. The VALUE clause is not allowed at the group level.

4. Format 2 is intended to provide a method of defining special character codes which are required for display or printer commands and for testing certain keyboard input characters such as function keys. Refer to your Commodore User's Guide relating to CHR$ codes.

For Example:

```
77 RETURN-CODE VALUE IS CHR 13 PIC X.
```

In the PROCEDURE DIVISION when a DISPLAY RETURN-CODE is executed the cursor will advance to the next line and then position to column 1.

5. Format 2 is an extension to the ANSI COBOL-74 standard.

WORKING-STORAGE SECTION

The WORKING-STORAGE SECTION is optional and is that part of the DATA DIVISION set aside for intermediate processing of data. The difference between the WORKING-STORAGE and FILE SECTIONS is that the former deals with data that is not associated with an input or output file. All clauses which are used in normal input or output record descriptions can be used in a WORKING-STORAGE record description.

WORKING-STORAGE STRUCTURE

Whereas the FILE SECTION is composed of file description (FD) entries and associated record description entries, the WORKING-STORAGE SECTION is composed only of record description entries and noncontiguous items. The WORKING-STORAGE SECTION begins with the section-header and a period, followed by data

description entries for noncontiguous WORKING-STORAGE items, and/or record description entries for WORKING-STORAGE records.

Each WORKING-STORAGE SECTION record name and noncontiguous item name must be unique.

General Example:

```
WORKING-STORAGE SECTION.
77    data-name-1

77    data-name-n
01    data-name-2
      02 data-name-3

01    data-name-4
      02 data-name-5
         03 data-name-6
```

NONCONTIGUOUS WORKING-STORAGE

Items in WORKING-STORAGE which have no hierarchical relationship to one another need not be grouped into records, provided they do not need to be further subdivided. These items are classified and defined as noncontiguous elementary items. Each of these items is defined in a separate data description entry which begins with the special level-number 77.

The following record description clauses are required in each entry:

Level-number 77
Data-name
The PICTURE clause or the USAGE IS INDEX clause.

The OCCURS clause is not meaningful on a 77 level item and will cause an error. Other data description clauses are optional and can be used to complete the description of the item if necessary.

WORKING-STORAGE RECORDS

Data elements and constants in WORKING-STORAGE which have a definite hierarchic relationship to one another must be grouped into records according to the rules for the formation of record descriptions. All clauses which are used in normal input or output record descriptions can be used in a WORKING-STORAGE description.

INITIAL VALUES

The initial value of any item in the WORKING-STORAGE SECTION except an index data item is specified by using the VALUE clause with the data item. The initial value of any index data item is unpredictable.

CODING THE WORKING-STORAGE SECTION

Figure 6-5 illustrates the coding of the WORKING-STORAGE SECTION

```
WORKING-STORAGE SECTION.
01  HDG-LINE.
    03  FILLER PIC X(52).
    03  DN2 PIC A(17) VALUE "SALES PERFORMANCE".

77  DISK-CONTROL     PICTURE 9(8).
77  TOTAL-SALES      PIC     9(11) VALUE 0.
77  SALES-QUOTA      PIC     9(10).
01  STATE-TABLE.

    05  STATE-KEY OCCURS 50.
        10  STATE-CODE  PIC 99.
        10  COUNTY      PIC  9.
        10  CITY        PIC  9.
```

SECTION 7 PROCEDURE DIVISION

GENERAL

The PROCEDURE DIVISION must be included in every COBOL 64 source program. This division must contain at least 1 paragraph.

A paragraph consists of a paragraph-name, followed by a period and a space, followed by zero, one, or more successive sentences. A paragraph ends immediately before the next paragraph-name or at the end of the PROCEDURE DIVISION.

A sentence consists of one or more statements and is terminated by a period.

A statement is a syntactically valid combination of words and symbols beginning with a COBOL 64 verb.

The term 'identifier' is defined as the word or words necessary to make unique reference to a data item.

EXECUTION OF THE PROCEDURE DIVISION

Execution begins with the first statement of the PROCEDURE DIVISION. Statements are executed in the order of appearance, except where the user indicates GO TO or PERFORM statements.

PROCEDURE DIVISION STRUCTURE

The PROCEDURE DIVISION is made up of the PROCEDURE DIVISION header and the PROCEDURE DIVISION body. Descriptions of these follow:

PROCEDURE DIVISION HEADER

The PRECEDURE DIVISION is identified by and must begin with the following header:

PROCEDURE DIVISION.

PROCEDURE DIVISION BODY

The body of the PROCEDURE DIVISION must conform to the following format.

```
{paragraph-name.  [sentence] ...}  ...
```

STATEMENTS

There are two types of statements; conditional statements and imperative statements.

CONDITIONAL STATEMENTS

A conditional statement specifies that the truth value of a condition is to be determined and that the subsequent action of the object program is dependent on this truth value.

A conditional statement is

1. An IF statement.
2. A READ statement that specifies the AT END or INVALID KEY phrase.
3. A WRITE statement that specifies the INVALID KEY phrase.
4. An arithmetic statement (ADD, DIVIDE, MULTIPLY, SUBTRACT) that specifies the SIZE ERROR phrase.
5. A FILTER-NUMERIC statement.

IMPERATIVE STATEMENTS

An imperative statement indicates a specific unconditional action to be taken by the object program. An imperative statement is any statement that is not a conditional statement. An imperative statement may consist of a sequence of imperative statements, each possibly separated from the next by a separator. The imperative verbs are:

```
ACCEPT
ACCEPT-1-KEY
ADD (1)
CLOSE
DEBUG-BREAK
DEBUG-TRACE-OFF
DEBUG-TRACE-ON
DISPLAY
DIVIDE (1)
EXIT
GO
MOVE
MULTIPLY (1)
OPEN
PERFORM
READ (2)
SET
STOP
SUBTRACT (1)
WRITE (3)
```

The numbers in parentheses following some of the verbs have the following meaning:

Number	Meaning
1	Without the optional `SIZE ERROR` phrase.
2	Without the optional `AT END` phrase or `INVALID KEY` phrase.
3	Without the optional `INVALID KEY` phrase.

When 'imperative-statement' appears in the general format of statements, it refers to a statement that begins with an imperative verb and specifies an unconditional action to be taken. An imperative statement may consist of a sequence of imperative statements. Imperative statements must be ended by a period, or an ELSE phrase associated with a previous IF statement.

RELATION CONDITION

A relation condition causes a comparison of two operands, each of which may be the data item referenced by an identifier or a literal. A relation condition has a truth value of TRUE if the relation exists between the operands. If either of the operands is a group item, the nonnumeric comparison rules apply.

General Format:

```
{identifier-1}   {IS [NOT] GREATER THAN}   {identifier-2}
{literal-1   }   {IS [NOT] LESS THAN   }   {literal-2   }
                 {IS [NOT] EQUAL TO    }
```

The first operand (identifier-1 or literal-1) is the subject of the condition; the second operand (identifier-2 or literal-2) is the object of the condition. The relation condition must contain at least one reference to a variable.

When used, NOT and the next key word are one relational operator that defines the comparison to be executed for truth value; for example, NOT EQUAL is a truth test for an unequal comparison; NOT GREATER is a truth test for an equal or less comparison. The meaning of the relational operators is as follows:

Relational Operator	Meaning
IS [NOT] GREATER THAN	Greater than or not greater than
IS [NOT] LESS THAN	Less than or not less than
IS [NOT] EQUAL TO	Equal to or not equal to

COMPARISON OF NUMERIC OPERANDS

For operands whose class is numeric, a comparison is made with respect to the algebraic value of the operands. The length of the literal in terms of number of digits represented, is not significant. Zero is considered a unique value regardless of the sign.

Unsigned numeric operands are considered positive for purposes of comparison.

COMPARISON OF NONNUMERIC OPERANDS

For nonnumeric operands, or one numeric and one nonnumeric operand, a comparison is made with respect to a specified collating sequence of characters. Refer to your Commodore User's Guide CHR$ codes for additional information. If one of the operands is specified as numeric, it must be an integer data item or an integer literal. The following conditions apply:

1. If the nonnumeric operand is an elementary data item or a nonnumeric literal, the numeric operand is treated as though it were moved to an elementary alphanumeric data item of the same size as the numeric data item (in terms of standard data format characters), and the contents of this alphanumeric data item are compared to the nonnumeric operand. Refer to the MOVE statement in this section.

2. If the nonnumeric operand is a group item, the numeric operand is treated as though it were moved to a group

item of the same size as the numeric data item (in terms of standard data format characters), and the contents of this group item are compared to the nonnumeric operand. Refer to the MOVE for additional information.

3. A noninteger numeric operand cannot be compared to a nonnumeric operand.

4. The operands must be the same size.

Comparison effectively proceeds by comparing characters in corresponding character positions starting from the high order end and continuing until either a pair of unequal characters is encountered or the low order end of the operand is reached, whichever comes first. The operands are determined to be equal if all pairs of characters compare equally through the last pair, when the low order end is reached.

The first encountered pair of unequal characters is compared to determine a relative position in the collating sequence. The operand that contains the character that is positioned higher in the collating sequence is considered to be the greater operand.

COMPARISONS INVOLVING INDEX-NAMES AND/OR INDEX DATA ITEMS

Relation tests may be made between:

1. Two index-names. The result is the same as if the corresponding occurrence numbers were compared.

2. An index-name and a data item (other than an index data item) or literal. The occurrence number that corresponds to the value of the index-name is compared to the data item or literal.

3. An index data item and an index-name or another index data item. The actual values are compared without conversion.

The comparison of an index data item with a literal or with any data item not specified above, is not allowed.

CLASS CONDITION

The class condition determines whether the operand is numeric, consisting entirely of the characters '0', '1', '2', '3', ..., '9', with or without the operational sign, or alphabetic, consisting entirely of the characters 'A', 'B', 'C', ..., 'Z', and space.

General Format:

```
identifier IS [NOT] { NUMERIC    }
                    { ALPHABETIC }
```

When used, NOT and the next key word specify one class condition that defines the class test to be executed for truth value; for example, NOT NUMERIC is a truth test for determining that an operand is nonnumeric.

The NUMERIC test cannot be used with an item whose data description describes the item as alphabetic or as a group item composed of elementary items whose data description indicates the presence of operational sign(s). If the data description of the item being tested does not indicate the presence of an operational sign, the item being tested is determined to be numeric only if the contents are numeric and an operationsl sign is not present. If the data description of the item does indicate the presence of an operational sign, the item being tested is determined to be numeric only if the contents are numeric and a valid operational sign is present.

The ALPHABETIC test cannot be used with an item whose data description describes the item as numeric. The item being tested is determined to be alphabetic only if the contents consist of any combination of the alphabetic characters 'A' through 'Z' and the space character.

COMMON PHRASES

In the statement descriptions that follow, several phrases appear frequently: the ROUNDED phrase and the SIZE ERROR phrase.

In the following discussion, a resultant-identifier is that identifier associated with a result of an arithmetic operation.

ROUNDED PHRASE

If, after decimal point alignment, the number of places in the fraction of the result of an arithmetic operation is greater than the number of places provided for the fraction of the resultant-identifier, truncation is relative to the size provided for the resultant-identifier. When rounding is requested, the absolute value of the resultant-identifier is increased by adding a one into the low-order digit whenever the absolute value of the next least significant digit of the intermediate data item is greater than or equal to five.

SIZE ERROR PHRASE

If, after decimal point alignment, the absolute value of a result exceeds the largest value that can be contained in the associated resultant-identifier, a size error condition exists. Division by zero always causes a size error condition. The size error condition applies only to the final results of an arithmetic operation and does not apply to intermediate results, except in the MULTIPLY and DIVIDE statements, in which case the size error condition applies to the intermediate results as well. If the ROUNDED phrase is specified, rounding takes place before checking for size error. When a size error condition occurs, the subsequent action depends on whether or not the SIZE ERROR phrase is specified.

1. If the SIZE ERROR phrase is not specified and a size error condition occurs, the resultant value is stored in each of the receiving fields left truncated where required. Values of resultant-identifier(s) for which no size error condition occurs are unaffected by size errors that occur for other resultant-identifier(s) during execution of this operation.

2. If the SIZE ERROR phrase is specified and a size error condition occurs, then the values of resultant-identifier(s) affected by the size errors are not altered. Values of resultant-identifier(s) size error condition occurs are unaffected by size errors that occur for other resultant-identifier(s) during execution of this operation. After completion of the execution of this operation, the imperative statement in the SIZE ERROR phrase is executed.

STATEMENT FORMATS

GENERAL RULES FOR STATEMENT FORMATS

The following paragraphs describe general rules for statement formats.

ARITHMETIC STATEMENTS

The arithmetic statements are ADD, DIVIDE, MULTIPLY, and SUBTRACT and have several common features:

1. The data descriptions of the operands need not be the same; any necessary conversion and decimal point alignment is supplied throughout the calculation.

2. The maximum size of each operand is 18 decimal digits.

3. Each arithmetic operation is evaluated using an intermediate data item for the result of the operation.

The contents of the intermediate data item are moved to the resultant-identifier according to the rules for the MOVE statement. Rounding is performed and the size error conditon is determined only during this MOVE operation.

INCOMPATIBLE DATA

Except for the class condition (refer to Class Condition in this section), when the contents of a data item are referenced in the PROCEDURE DIVISION and the contents of that data item are not compatible with the class specified for that data item by the PICTURE clause, then the result of such a reference is undefined.

SPECIFIC VERB FORMATS

The specific verb formats, together with a detailed discussion of the restrictions and limitations associated with each, appear on the following pages in alphabetic sequence.

ACCEPT

The ACCEPT statement is used to input data from the keyboard and placed in the specified data item.

General Format:

```
ACCEPT identifier
```

Syntax Rules:

1. If the identifier describes a numeric item it must be an integer.

General Rules:

1. The ACCEPT statement causes the transfer of data from the keyboard. This data replaces the contents of the data item named by the identifier.

2. The maximum number of characters that can be transferred is 80. The RETURN key terminates the transfer.

3. The ACCEPT statement causes the information requested to be transferred to the data item specified by identifier according to the rules of the MOVE statement.

4. As each key is entered it is displayed on the screen at the current cursor position. The cursor control and insert/delete keys are active.

ACCEPT-1-KEY

The ACCEPT-1-KEY statement is used to input 1 character from the keyboard and place it in the specified data item. This verb is an extension to ANSI COBOL-74.

General Format:

```
ACCEPT-1-KEY identifier
```

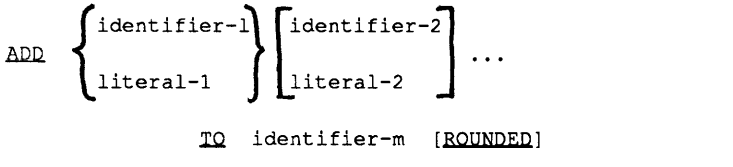
This statement differs from the ACCEPT statement in that it will enable any one key on the keyboard including function keys. Refer to your Commodore User's Guide under CHR$ codes for each key's definition. The key entered is not displayed.

ADD

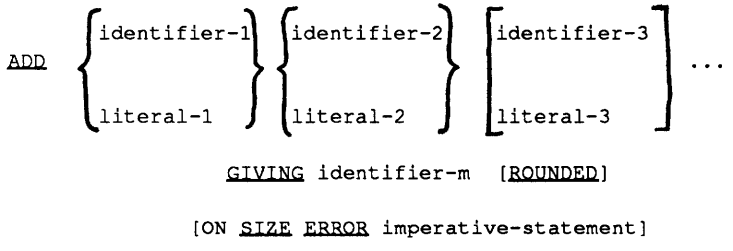
The ADD statement causes two or more numeric operands to be summed and the result to be stored.

General Format:

Format 1:



Format 2:



Syntax Rules:

1. In formats 1 and 2, each identifier must refer to an elementary numeric item, except that in Format 2 the identifier following the word GIVING must refer to either an elementary numeric item or an elementary numeric edited item.

2. Each literal must be a numeric literal.

General Rules:

1. Additional rules and explanation relative to this statement are given in the appropriate paragraphs. Refer to ROUNDED Phrase, SIZE ERROR Phrase, Arithmetic Statements.

2. If Format 1 is used, the values of the operands preceding the word TO are added together, then the sum is added to the current value of identifier-n storing the result immediately into identifier-n.

3. If Format 2 is used, the values of the operands preceding the word GIVING are added together, then the sum is stored as the new value of identifier-m, the resultant-identifier.

4. The system ensures that enough places are carried so that significant digits are not lost during execution.

Examples:

Assume as initial values for each ADD: X=2, Y=10, Z=15, TOT=50.

Format 1:

```
ADD X TO TOT.                Results TOT=52

ADD X  Y  Z TO TOT           Results TOT=77
```

Format 2:

```
ADD X  Y GIVING TOT.         Results TOT=12

ADD X  Y  Z GIVING TOT       Results TOT=27
```

CLOSE

General Format:

```
CLOSE  file-name
```

The CLOSE statement terminates the processing of a file.

General Rules:

A CLOSE statement may only be executed for a file in an open mode. Refer to I-O status under the File Concepts section.

It is very important that you CLOSE files once you have finished using them. Closing a disk file causes the system to properly allocate space and update the directory. If you do not CLOSE the disk file, all of your data will be lost.

DEBUG-BREAK

This verb will cause an execution break if the program is executing in the DEBUG Mode. Refer to the DEBUG Mode sections. The verb is ignored if not executing in the DEBUG Mode. This verb is an extension to the ANSI standard.

DEBUG-TRACE-OFF

When executing in the DEBUG Mode this verb will turn off the trace feature. Refer to the DEBUG section. This verb is ignored if executing in the DEBUG Mode. This verb is an extension to the ANSI standard.

DEBUG-TRACE-ON

This verb will cause the trace feature to be turned on if executing in the DEBUG Mode. Refer to the DEBUG Mode section. The verb is ignored if not executing in the DEBUG Mode. This verb is an extension to the ANSI standard.

DISPLAY

The DISPLAY statement causes the data items to be displayed on the screen.

General Format:

```
DISPLAY { identifier-1 } [ identifier-2 ] ...
        { literal-1    }  [ literal-2    ]
```

Syntax Rules:

If the literal is numeric, then it must be an unsigned integer.

General Rules:

1. The DISPLAY statement causes the contents of each operand to be transferred to the hardware device in the order listed.

2. The maximum number of characters that can be transmitted is unlimited.

3. When a DISPLAY statement contains more than one operand, the values of the operands are transferred in the sequence in which the operands are encountered.

4. For any one data item, if a character code 13 (RETURN) is encountered the transfer will be terminated after sending the RETURN code.

5. It is not recommended to DISPLAY an identifier which is defined as signed numeric. This is due to the fact that the sign character is combined with the most significant number of the data item.

6. The display begins at the current cursor location.

DIVIDE

The DIVIDE statement divides one numeric data item into others and sets the values of data items equal to the quotient.

General Formats:

Format 1:

```
DIVIDE { identifier-1 } INTO identifier-2 [ROUNDED]
       { literal-1    }

    [ON SIZE ERROR imperative-statement]
```

Format 2:

```
DIVIDE { identifier-1 } { BY   } { identifier-2 }
       { literal-1    } { INTO } { literal-2    }

          GIVING identifier-3 [ROUNDED]

    [ON SIZE ERROR imperative-statement]
```

Syntax Rules:

1. Each identifier must refer to an elementary numeric item, except that the identifier associated with the GIVING phrase must refer to either an elementary numeric item or an elementary numeric edited item.

2. Each literal must be a numeric literal.

General Rules:

1. Additional rules and explanations relative to this statement are given in the appropriate paragraphs. Refer to Arithmetic Statements, ROUNDED phrase and the SIZE ERROR phrase.

2. When Format 1 is used, the value of identifier-2 is divided by either the value of identifier-1 or literal-1. The value of the dividend (identifier-2) is replaced by this quotient.

3. When Format 2 is used, the value of identifier-1 or literal-1 is divided by the value of identifier-2 or literal-2, and the result is stored in identifier-3.

EXIT

The EXIT statement provides a means of documenting the logical end point for a series of paragraphs that may be executed under the control of a PERFORM statement.

General Format:

```
EXIT.
```

Syntax Rules:

1. The EXIT statement must appear in a sentence alone.
2. The EXIT sentence must be the only sentence in the paragraph.

General Rules:

1. An EXIT statement serves only to enable the user to assign a procedure-name to a given point in a program. Such an EXIT statement has no other effect on the execution of the program.

FILTER-NUMERIC

The FILTER-NUMERIC statement filters and validates alphanumeric data to numeric data format. This verb is an extension to ANSI COBOL-74.

General Format:

```
FILTER-NUMERIC  identifier-1 TO identifier-2

ON ERROR  imperative-statement
```

Syntax Rules:

1. Identifier-1 represents the sending area and identifier-2 represents the receiving area.

2. Identifier-2 must be defined as a numeric data item.

General Rules:

1. The contents of identifier-1 are examined. Valid characters are 0 through 9 and the decimal-point. Leading and trailing space characters are also valid. Only one decimal point character is allowed and only if there is one or more decimal places in the receiving fields picture. A decimal point without other characters is invalid.

2. If the above tests are passed the size of significant data is evaluated against the size of identifier-2. If there is no size problem, including decimal alignment, the data is transferred to identifier-2.

3. If any of the above tests fail the `ON ERROR` imperative statement is processed.

4. The `DECIMAL POINT IS COMMA` clause apples to this statement.

This verb is intended to process data received from the keyboard (`ACCEPT`) or other systems.

Examples: The receiving numeric PICTURE is 999V99.

Sending Data	Result
123.45	12345
1	00100
(all spaces)	00000
.12	00012
1 2	ERROR-embedded space
1234.5	ERROR-size
1.2345	ERROR - size
1.2.3	ERROR - more than 1 decimal point
1A2	ERROR-invalid character
.	ERROR-decimal point only

GO TO

The GO TO statement causes control to be transferred from one part of the PROCEDURE DIVISION to another.

General Format:

Format 1:

```
GO TO paragraph-name-1
```

Format 2:

```
GO TO paragraph-name-1 [paragraph-name-2] ... paragraph-name-n

     DEPENDING ON identifier
```

Syntax Rules:

1. Identifier is the name of a numeric elementary item described without any positions to the right of the assumed decimal point.

2. If a GO TO statement, represented by Format 1, appears in a consecutive sequence of imperative statements within a sentence, it must appear as the last statement in that sequence.

General Rules:

1. When a GO TO statement, represented by Format 1 is executed, control is transferred to paragraph-name-1.

2. When a Format 2 GO TO statement is executed, control is transferred to the paragraph-name whose ordinal position in the list following the GO TO corresponds to the value of the identifier being 1, 2, ..., n. If the value of the identifier is anything other than the positive or unsigned integers 1, 2, ..., n, then no transfer occurs and control passes to the next statement in the normal sequence for execution.

IF

The IF statement causes a condition to be evaluated. The subsequent action of the object program depends on whether the value of the condition is TRUE or FALSE.

General Format:

```
IF condition [statement-1      ] [ELSE statement-2      ]
             [NEXT SENTENCE    ] [ELSE NEXT SENTENCE    ]
```

Syntax Rules:

1. Statement-1 and statement-2 must represent an imperative statement.

2. The ELSE NEXT SENTENCE phrase may be omitted if it immediately precedes the terminal period of the sentence.

General Rules:

1. When an IF statement is executed, the following transfers of control occur:

   a. If the condition is TRUE, statement-1 is executed, if specified. If statement-1 contains a procedure branching statement, control is explicitly transferred in accordance with the rules of that statement. Refer to Categories of Statements in this section. If statement-1 does not contain a procedure branching statement, the ELSE phrase, if specified, is ignored and control passes to the next executable sentence.

   b. If the condition is TRUE and the NEXT SENTENCE phrase is specified instead of statement-1, the ELSE phrase, if specified, is ignored and control passes to the next executable sentence.

c. If the condition is FALSE, statement-1 or NEXT SENTENCE is ignored, and statement-2 if specified, is executed. If statement-2 contains a procedure branching statement, control is explicitly transferred in accordance with the rules of that statement. Refer to Categories of Statements in this section. If statement-2 does not contain a procedure branching statement, control passes to the next executable sentence. If the ELSE statement-2 phrase is not specified, statement-1 is ignored and control passes to the next executable sentence.

d. If the condition is FALSE, and the ELSE NEXT SENTENCE phrase is specified, statement-1 is ignored, if specified, and control passes to the next executable sentence.

Examples:

```
IF APPLE IS EQUAL TO RED
     DISPLAY "GOOD"
     PERFORM GOOD-APPLE
ELSE PERFORM BAD-APPLE.


IF APPLE EQUAL RED
     NEXT SENTENCE
          ELSE DISPLAY "BAD"

          ADD 1 TO TOT-BAD-APPLES.

IF APPLE NOT EQUAL RED
     ADD 1 TO TOT-BAD-APPLES.
```

MOVE

The MOVE statement transfers data, in accordance with the rules of editing, to one or more data areas.

General Format:

```
MOVE { identifier-1 } TO identifier-2 [identifier-3] ...
     { literal      }
```

Syntax Rules:

1. Identifier-1 and literal represent the sending area; identifier-2, identifier-3,..., represent the receiving area.

2. An index data item cannot appear as an operand of a MOVE statement. Refer to the USAGE clause in Section 6.

General Rules:

1. The data designated by the literal or identifier-1 is moved first to identifier-2, then to identifier-3, and so on. The rules governing identifier-2 also apply to the other receiving areas. Any subscripting or indexing associated with identifier-2 is evaluated immediately before the data is moved to the respective data item.

   Any subscripting or indexing associated with identifier-1 is evaluated only once, immediately before data is moved to the first of the receiving operands. The result of the statement:

   ```
   MOVE a (b) TO b  c (b)
   ```

   is equivalent to:

   ```
   MOVE a (b) TO temp
   MOVE temp TO b
   MOVE temp TO c (b)
   ```

Temp is an intermediate result item provided by the system.

2. Any MOVE in which the sending and receiving items are both elementary items is an elementary move. Every elementary item belongs to one of the following categories: numeric, alphabetic, alphanumeric, numeric edited, alphanumeric edited. These categories are described in the PICTURE clause in Section 6. Numeric literals belong to the category numeric, and nonnumeric literals belong to the category alphanumeric.

   The following rules apply to an elementary move between the categories:

   a. A numeric edited, alphanumeric edited, or alphabetic data item must not be moved to a numeric or numeric edited data item.

   b. A numeric literal, a numeric data item or a numeric edited data item must not be moved to an alphabetic data item.

   c. A noninteger numeric literal or a noninteger numeric data item must not be moved to an alphanumeric or alphanumeric edited data item.

   d. All other elementary moves are legal and are performed according to the rules given in General Rule 3.

3. Any necessary conversion of data from one form of internal representation to another takes place during legal elementary moves, along with any editing specified for the receiving data item:

   a. When an alphanumeric edited or alphanumeric item is a receiving item, alignment and any necessary space filling takes place as defined under Standard Alignment Rules in Section 2. If the size of the sending item is greater than the size of the receiving item, the excess characters are

truncated on the right after the receiving item is filled. If the sending item is described as being signed numeric, the operational sign is not moved.

b. When a numeric or numeric edited item is the receiving item, alignment by decimal point and any necessary zero-filling takes place as defined under the Standard Alignment Rules, except where zeroes are replaced because of editing requirements.

   1) When a signed numeric item is the receiving item, the sign of the sending item is placed in the receiving item. If the sending item is unsigned, a positive sign is generated for the receiving item.

   2) When an unsigned numeric item is the receiving item, the absolute value of the sending item is moved and no operational sign is generated for the receiving item.

   3) When a data item described as alphanumeric is the sending item, data is moved as if the sending item were described as an unsigned numeric integer.

c. When a receiving field is described as alphabetic, justification and any necessary space-filling takes place as defined under the Standard Alignment Rules. If the size of the sending item is greater than the size of the receiving item, the excess characters are truncated on the right after the receiving item is filled.

5. Any move that is not an elementary move is treated exactly as if it were an alphanumeric to alphanumeric elementary move, except that there is no conversion of data from one form of internal representation to another. In such a move, the receiving area is filled without consideration for the individual elementary or group items contained within either the sending or receiving area.

6. The validity of the various types of MOVE statements is summarized in Table 7-4.

Table 7-4. A Valid MOVE Statement

Category of Sending Data Item	Category of Receiving Data Item: Alphabetic	Alphanumeric Edited / Alphanumeric	Numeric Integer Numeric	Numeric Noninteger Numeric Edited
ALPHABETIC	YES	YES		NO
ALPHANUMERIC	YES	YES		YES
ALPHANUMERIC EDITED	YES	YES		NO
NUMERIC INTEGER	NO	YES		YES
NUMERIC NONINTEGER	NO	NO		YES
NUMERIC EDITED	NO	YES		NO

MULTIPLY

The MULTIPLY statement causes numeric data items to be multiplied and sets the value of a data item equal to the result.

General Format:

Format 1:

```
MULTIPLY {identifier-1} BY identifier-2 [ROUNDED]
         {literal-1   }

  [ON SIZE ERROR imperative-statement]
```

Format 2:

```
MULTIPLY {identifier-1} BY {identifier-2} GIVING identifier-3
         {literal-1   }    {literal-2   }

  [ROUNDED]

  [ON SIZE ERROR imperative-statement]
```

Syntax Rules:

1. Each identifier must refer to a numeric elementary item, except that in Format 2 the identifier following the word GIVING must refer to either an elementary numeric item or an elementary numeric edited item.

2. Each literal must be a numeric literal.

3. The composite of operands, determined by superimposing all receiving data items of a given statement, aligned on decimal points, must not exceed 18 digits.

General Rules:

1. Refer to ROUNDED Phrase, SIZE ERROR Phrase in Arithmetic Statements in this section for additional rules and information.

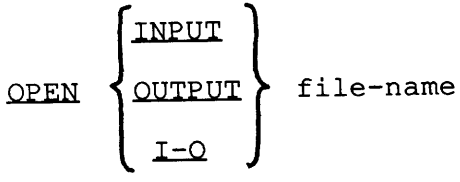
2. When Format 1 is used, the value of identifier-1 or literal-1 is multiplied by the value of identifier-2. The value of the multiplier (identifier-2) is replaced by this product.

3. When Format 2 is used, the value of identifier-1 or literal-1 is multiplied by identifier-2 or literal-2 and the result is stored in identifier-3.

OPEN

The OPEN statement initiates the processing of files. It also performs checking of labels and other operations.

General Format:



Syntax Rules:

1. The I-O phrase can be used only for disk files. The disk file must be defined with ORGANIZATION IS RELATIVE and ACCESS MODE IS RANDOM clauses.

2. OPEN INPUT file-name must not be a printer file.

General Rules:

1. The successful execution of an OPEN statement determines the availability of the file and results in the file being in an open mode.

2. The execution of an OPEN statement does not affect either the contents or availability of the file's record area.

3. When a given file is not in an open mode, no statement that references that file can be executed successfully.

4. A file may be opened with the INPUT, OUTPUT and I-O phrases in the same program. Following the initial execution of an OPEN statement for a file, each subsequent OPEN statement execution for that same file must be preceded by the execution of a CLOSE statement for that file.

5. Execution of the OPEN statement does not obtain or release the first data record.

6. The beginning labels are processed as follows:

   a. When the INPUT phrase is specified, the execution of the OPEN statement causes the labels to be checked in accordance with conventions for input label checking.

   b. When the OUTPUT phrase is specified, the execution of the OPEN statement causes the labels to be written in accordance with conventions for output label writing.

7. The file description entry for the file-name must be equivalent to that used when this file was created.

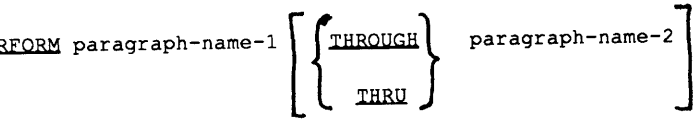
8. For files being opened with the INPUT phrase, the OPEN statement sets the current record pointer to the first record currently existing within the file.

9. Upon successful execution of an OPEN statement with the OUTPUT phrase specified, a file is created. At that time the associated file contains no data records.

PERFORM

The PERFORM statement is used to transfer control explicitly to one or more paragraphs and to return control implicitly whenever execution of the specified paragraph is complete

General Format:



Syntax Rules:

1. The words THRU and THROUGH are equivalent.

General Rules:

1. When the PERFORM statement is executed, control is transferred to the first statement of the paragraph named paragraph-name-1. This transfer of control occurs only once for each execution of a PERFORM statement. An implicit transfer of control to the next executable statement following the PERFORM statement is established as follows:

   a. IIf paragraph-name-2 is not specified, then the return is after the last statement of paragraph-name-1.

   b. If paragraph-name-2 is specified, then the return is after the last statement of the paragraph-name-2.

2. No particular sequential relationship is required to exist between paragraph-name-1 and paragraph-name-2. There may be more than one logical path of program

control through the performed range of paragraphs. A common method, though not a required one, of documenting the terminal paragraph of a performed range of paragraphs is through the use of the EXIT statement.

3. An implicit return mechanism is established at the end of a performed range of paragraphs and is activated by the execution of a PERFORM statement. Program control reaching an active return mechanism always returns to the activating PERFORM statement. A return mechanism permanently deactivates by transferring program control back to a PERFORM statement. An active return mechanism is temporarily deactivated by the execution of a PERFORM statement. Program control always passes through a nonactive return mechanism to the next executable statement following the PERFORM range.

4. A paragraph executed under the control of a PERFORM statement may execute PERFORM statements. There is no requirement that the range of paragraphs executed under the control of the nested PERFORM statement be declared totally within, or disjoint from, the range of paragraphs executed by the first PERFORM statement. The permanent deactivation of an active return mechanism causes the last return mechanism temporarily deactivated to again become active, allowing overlapping PERFORM ranges, or two or more PERFORM ranges that have a common exit point, to logically execute the same as disjoint PERFORM ranges.

Transferring program control, by means of a GO TO statement, from a range of paragraphs being executed under control of a PERFORM statement does not cause the return mechanism to be deactivated. This is allowed but this is not considered good programming practice and should be avoided! Subsequently, transferring program control back into the PERFORM range causes control to return to the PERFORM statement, provided that the return

mechanism is still active. Repeatedly branching from a PERFORM range without allowing control to ever reach an active return mechanism may cause the program to terminate abnormally by exhausting the resources allocated to account for return mechanisms. In such a case, the error message PERFORM STACK ERROR is displayed

Example:

```
START.   PERFORM PARA
         PERFORM PARA THRU PARC.
ENDIT.   STOP RUN.
PARA.    ADD....
PARB.    MOVE....
PARC.    PERFORM PARB
```

The execution sequence would be:

```
START
     PARA
     PARA
     PARB
     PARC
          PARB
ENDIT
```

READ

For sequential access, the READ statement makes available the next logical record from a file. For random access, the READ statement makes available a specified record.

General Format:

Format 1:

```
READ file-name RECORD

  AT END imperative-statement
```

Format 2:

```
READ file-name RECORD

  INVALID KEY imperative-statement
```

Syntax Rules:

1. Format 1 must be used for all files in sequential access mode.

2. Format 2 is used for files in random access mode.

General Rules:

1. The associated file must be open in the INPUT or I-O mode. Refer to the OPEN statement in this section.

2. The execution of the READ statement causes the value of the FILE STATUS data item, if any, associated with file-name to be updated. Refer to I-O Status in Section 5.

3. If, at the time of the execution of a Format 1 READ statement, no next logical record exists in the file, the AT END condition occurs, and the execution of the

READ statement is considered unsuccessful. Refer to I-O Status in Section 5.

4. When the AT END condition is recognized, the following actions are taken in the specified order:

   a. A value is placed into the FILE STATUS data item, if specified for this file, to indicate an AT END condition. Refer to I-O Status in Section 5.

   b. Control is transferred to the AT END imperative statement.

   c. The execution of the input-output statement which caused the condition is unsuccessful.

5. Following the unsuccessful execution of any READ statement, the contents of the associated record area and the position of the current record pointer are undefined.

6. When the AT END condition has been recognized, a Format 1 READ statement for that file must not be executed without first executing a successful CLOSE statement followed by the execution of a successful OPEN statement for that file.

7. In a Relative File with access mode sequential declared, if the RELATIVE KEY phrase is specified, the execution of a Format 1 READ statement updates the contents of the RELATIVE KEY data item so that it contains the relative record number of the record made available.

8. For a Relative File with access mode random declared, the execution of a Format 2 READ statement sets the current record pointer and makes available the record whose relative record number is contained in the data item named in the RELATIVE KEY phrase for the file. If the file does not contain such a record, the INVALID KEY condition exists and execution of the

READ statement is unsuccessful. Refer to the INVALID KEY condition under Invalid Key in Section 5.

SET

The SET statement establishes reference points for table handling operations by setting index-names associated with table elements.

General Format:

```
SET  { identifier-1 }      TO  { identifier-3 }
     { index-name-1 }          { index-name-3 }
                               { integer-1    }
```

Syntax Rules:

1. Integer-1 may be signed but must be plus.

General Rules:

1. Index-names are considered related to a given table and are defined by being specified in the INDEXED BY clause.

2. If index-name-3 is specified, the value of the index before the execution of the SET statement should correspond to an occurrence number of an element in the associated table.

   If index-name-1 is specified, the value of the index after the execution of the SET statement should correspond to an occurrence number of an element in the associated table.

   When a statement using the index-name to refer to a table element is executed, the value in the index or the value produced by relative indexing must fall within

the range specified by the OCCURS clause defining the table. Otherwise, an abnormal termination of the program occurs. Refer to Indexing in Section 2.

3. When a SET statement is executed, the following actions occur:

   a. Index-name-1 is set to a value causing it to refer to the table element that corresponds in occurrence number to the table element referenced by index-name-3, identifier-3, or integer-1. If identifier-3 is an index data item, or if index-name-3 is related to the same table as index-name-1, no conversion takes place.

   b. If identifier-1 is an index data item, it may be set equal to either the contents of index-name-3 or identifier-3, where identifier-3 is also an index data item. No conversion takes place in either case.

   c. If identifier-1 is not an index data item, it may be set only to an occurrence number that corresponds to the value of index-name-3. Neither identifier-3 nor integer-1 can be used in this case.

4. Data in Table 7-6 represents the validity of various operand combinations in the SET statement. The general rule reference (for example, 3b) indicates the applicable general rule.

Table 7-6. SET Statement Combinations

Sending Item	Receiving Item		
	Integer Data Item	Index-Name	Index Data Item
Integer Literal	No/3c	Valid/3a	No/3b
Integer Data Item	NO/3c	Valid/3a	No/3b
Index-Name	Valid/3c	Valid/3a	Valid/3b*
Index Data Item	No/3c	Valid/3a*	Valid/3b*

* No conversion takes place *

STOP

The STOP statement causes a permanent suspension of the execution of the object program.

General Format:

STOP RUN

Syntax Rules:

1. If a STOP RUN statement appears in a consecutive sequence of imperative statements within a sentence, it must appear as the last statement in that sequence.

General Rules:

1. The ending procedure established by the COBOL 64 system is instituted.

SUBTRACT

The SUBTRACT statement is used to subtract one or the sum of two or more numeric data items from one item and set the value of one item equal to the result.

General Format:

Format 1:

```
SUBTRACT { identifier-1 } [ identifier-2 ] ...FROM identifier-m
         { literal-1    } [ literal-2    ]

    [ROUNDED]

    [ON SIZE ERROR imperative-statement]
```

Format 2:

```
SUBTRACT { identifier-1 } [ identifier-2 ] ...FROM { identifier-m }
         { literal-1    } [ literal-2    ]         { literal-m    }

    GIVING identifier-n

    [ROUNDED]

    [ON SIZE ERROR imperative-statement]
```

Syntax Rules:

1. Each identifier must refer to a numeric elementary item except that in Format 2, the identifier following the word GIVING must refer to either an elementary numeric item or to an elementary numeric edited item.

2. Each literal must be a numeric literal.

General Rules:

1. Additional rules and explanations related to this statement are given in the appropriate paragraphs. Refer to ROUNDED Phrase, SIZE ERROR Phrase in Arithmetic Statements in this section.

2. In Format 1, all literals or identifiers preceding the word FROM are added together, and this total is subtracted from the current value of identifier-m. The result is immediately stored into identifier-m.

3. In Format 2, all literals or identifiers preceding the word FROM are added together, the sum is subtracted from literal-m or identifier-m, and the result of the subtraction is stored as the new value of identifier-m.

Examples:

Assume as initial values for each subtract;
X=2, Y=10, Z=15, TOT=50, and SUB=30.

Format 1:

```
SUBTRACT X FROM TOT.          results TOT=48
SUBTRACT X Y Z FROM TOT       results TOT=23
```

Format 2:

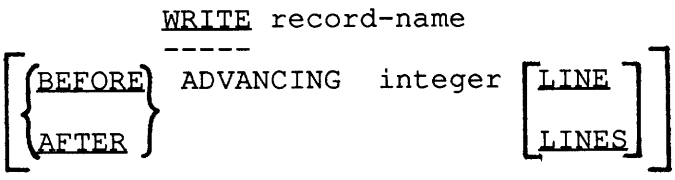
```
SUBTRACT X Y FROM SUB GIVING TOT.    results TOT=18
SUBTRACT X Y FROM Z GIVING TOT       results TOT=3
```

WRITE

The WRITE statement releases a logical record for an output file. It can also be used for vertical positioning of lines for a printer.

General Format:

Format 1:



Format 2:

```
WRITE record-name

INVALID KEY imperative-statement
```

Syntax Rules:

1. The record-name is the name of a logical record in the FILE SECTION of the DATA DIVISION.
2. Integer-1 may not be zero.
3. Format 2 is used for Organization Relative Files.

General Rules:

1. The associated file must be open in the OUTPUT or I-O mode at the time of the execution of this statement.
2. The execution of a WRITE statement has no effect upon either the contents or accessibility of the record area.

3. The execution of the WRITE statement causes the value of the FILE STATUS data item, if any, associated with the file to be updated. Refer to I-O Status in Section 5.

4. The maximum record size for a file is established when the file is created and must not subsequently be changed.

5. The number of character positions on a disk storage device required to store a logical record in a file may or may not be equal to the number of character positions defined by the logical description of that record in the program.

6. The execution of the WRITE statement releases a logical record to the operating system.

PRINTER FILES

1. The ADVANCING phrase allows control of the vertical positioning of each line on a printed page. If the ADVANCING phrase is not used, automatic advancing is provided to act as if the user had specified AFTER ADVANCING 1 LINE. If the ADVANCING phrase is used, advancing is provided as follows:

   a. IIf integer-1 is specified, the page is advanced the number of lines equal to the value of integer-1.

   b. If the BEFORE phrase is used, the line is written before the page is advanced.

2. During the transfer of data to the printer, if a character code 13 (RETURN) is encountered the transfer is terminated after sending the RETURN code.

DISK FILES

SEQUENTIAL FILES:

1. When an attempt is made to write beyond the externally defined boundaries of a Sequential File, an exception condition exists and the contents of the record area are unaffected. The value of the FILE STATUS data item, if any, of the associated file is set to a value indicating a boundary violation. Refer to I-O Status in Section 5.

RELATIVE FILES:

1. When a Relative File is opened in the output mode, records may be placed into the file in one of the following ways:

   a. If the access mode is sequential, the WRITE statement causes a record to be released. The first record has a relative record number of 1 and subsequent records released have relative record numbers of 2, 3, 4, and so on. If the RELATIVE KEY data item has been specified in the file control entry for the associated file, the relative record number of the record just released is placed into the RELATIVE KEY data item during execution of the WRITE statement.

   b. If the access mode is random, before the execution of the WRITE statement, the value of the RELATIVE KEY data item must be initialized in the program with the relative record number to be associated with the record in the record area. That record is then released by execution of the WRITE statement.

2. When a Relative File is opened in the I-O mode and the access mode is random, records are to be inserted in the associated file. The value of the RELATIVE KEY data item must be initialized by the program with the relative record number to be associated with the record

in the record area. Execution of a WRITE statement then causes the contents of the record area to be released.

3. The INVALID KEY condition exists when an attempt is made to write beyond the externally defined boundaries of the file.

CHAPTER 7

START UP/OPERATING INSTRUCTIONS

Connect the system as described in your Commodore User's Guide. Now turn on the equipment in the following order:

1. Printer (if present)
2. Computer
3. Disk drives
4. TV or monitor

Insert the COBOL 64 diskette and type:

```
LOAD "COBOL 64",8,1
```

followed by the RETURN key. The system loading process will then take place. When completed, READY will appear on the screen. Now type:

```
SYS 2051
```

followed by the RETURN key. The COBOL 64 System will then start.

CHAPTER 8

MAIN MENU

The COBOL 64 Main Menu is displayed on the screen and contains the following functions:

```
1=EDIT

2=RUN

3=DEBUG

4=SAVE

5=GET

6=NEW-PROG/EDIT

7=CRUNCH

8=PRINT-ON

9=PRINT-OFF

10=NEW-NAME
```

At this point enter the number of the function desired followed by the RETURN key.

The following sections will describe each function in detail. They appear in alphabetical order for easy reference.

Main Menu

CRUNCH

The CRUNCH function is used to reduce the memory size of your program file. A file must be present, refer to GET and NEW-PROG. If the COEDIT program overlay is not present in memory, it will be automatically loaded by the system from disk. Messages will appear on the screen when this occurs.

While developing your program, each time you delete or change lines the memory space for the old lines is no longer available for new lines. This is not a problem unless you exceed the maximum available memory. At this time or at any time you wish to consolidate memory, invoke this function.

The processing associated with this function includes writing to disk a temporary file with the prefix CS (COBOL sequential) before your file name. eg. CSYOUR-NAME. A warning message will appear on the screen which will allow you to remove the COBOL 64 diskette and insert your diskette. Enter any key on the keyboard to continue. Once the temporary file is written, it is then read back into the system. As each line is written to disk or read back from disk, it will appear on the screen. At the completion of this task a warning message will appear which allows you to save the new consolidated program on disk. The system will now return to the Main Menu.

Main Menu

DEBUG

This function directs the system to begin executing the current program file in memory. A file must be present, refer to GET or NEW-PROG. The DEBUG function is similar to the RUN function described below. In addition to executing your program a number of powerful debug features are provided which are intended to facilitate the debugging of your program. All debugging in COBOL 64 is accomplished at the source (symbolic) language level. There is no need to be concerned with machine language, memory addressing or hexadecimal notation.

Before debugging can begin, your program is tested to determine if it has been successfully syntaxed. If it has, the CORD (Run/Debug) program overlay is automatically loaded from disk if required. If your program had not previously been syntaxed the syntax process will begin. Refer to EDIT SYNTAX for additional details. If the syntax process is unsuccessful the system resumes at the main menu. If successful the system proceeds with the CORD program overlay.

At the beginning of DEBUG Mode a "START DEBUG" message will appear on the screen followed by the DEBUG feature menu. At this time you may need to remove the COBOL 64 diskette and insert your diskette if your program is going to use the disk drive.

The following is a list of the DEBUG Menu features. Each one is described in detail below. Enter the feature number desired followed by the RETURN key.

```
 1=START-PROG
 2=RESUME-PROG
 3=SINGLE-ON
 4=SINGLE-OFF
 5=EXIT
 6=BREAK1
 7=BREAK2
 8=BREAK3
 9=OPTIONS
10=TRACE-ON-LINE
11=TRACE-OFF-LINE
12=TRACE-FAST
13=TRACE-SLOW
14=TRACE-ON
15=TRACE-OFF
16=RESET-OPTIONS
```

START-PROG

This selection will cause your program to begin executing at the first PROCEDURE DIVISION statement in your program. Your program is initialized with its starting VALUE clauses as required. To simplify debugging, all other data items are initialized to the numeral 9. During execution, if your program references a data item which you did not properly initialize, the 9's will be obvious; otherwise you would see strange characters on the screen for a DISPLAY statement or as a function of the trace feature.

RESUME-PROG

This selection allows you to continue execution from the point where it was before entering the debug menu state.

This selection cannot be used when starting a program, or after a STOP RUN verb, you must use the START-PROG.

SINGLE-ON

This selection turns on the "single step" feature. Single stepping allows you to step through your program execution one statement at a time. Once your program begins executing (see START-PROG or RESUME-PROG) each statement will display S=NNNNNN where N is equal to the line number of the statement followed by the statement. At this point depress the RETURN key to execute the next statement. Any other key will direct the system to the DEBUG Menu. This feature will also automatically turn on the trace feature, refer to TRACE-ON. Comment lines are ignored during execution.

SINGLE-OFF

The single step feature described above is turned off. The trace feature is also turned off.

EXIT

This selection will direct the system to exit the DEBUG Mode and proceed to the COBOL 64 Main Menu.

BREAK1 BREAK2 BREAK3

A selection of the break feature allows you to enter a statement line number which when executed will cause the system to enter the DEBUG Menu state. Note that this occurs before execution of the statement in the selected line number. The screen will contain a B=NNNNNN where N is equal to the line number. The system provides for one to three line numbers plus the verb DEBUG-BREAK which you can place in your program as required. Comment lines are ignored during execution.

OPTIONS

The selection of this feature directs the system to display the current state of all DEBUG Menu options.

TRACE-ON-LINE

A selection of this feature allows you to enter a statement line number which when executed will cause the system to turn on the trace feature. Refer to TRACE-ON for additional information.

TRACE-OFF-LINE

A selection of this feature allows you to enter a statement line number which when executed will cause the system to turn off the trace feature. Refer to TRACE-ON and TRACE-OFF for more details.

TRACE-FAST

The TRACE-FAST feature displays all trace information at full speed. This is the default setting for the trace feature.

TRACE-SLOW

The TRACE-SLOW feature provides for slowing down the speed of the trace display such that it is more readable during execution.

TRACE-ON

The selection of the TRACE-ON feature provides for information to be displayed during execution of your program. The TRACE-ON feature is a default setting.

As each statement is executed the system displays T=NNNNNN followed by the statement; where N is equal to the statement line number. All comment lines are ignored during tracing. If the statement being traced has a receiving data item such as MOVE A TO B then C= is displayed followed by the new contents of the data item. The size of the display is limited to 18 characters. If the receiving field is a numeric data item the contents display is enhanced to include the sign (+-) if present and the letter 'V' in the assumed decimal point position.

Example:

The picture of A is equal to S99V99.

```
T=000100 MOVE +1.2 TO A

C=+01V20
```

In addition to the TRACE-ON and TRACE-ON-LINE features, the system provides a DEBUG-TRACE-ON verb which you can insert in your program as required.

TRACE-OFF

The trace feature described above is turned off. There is also a DEBUG-TRACE-OFF verb for this purpose which you can insert in your program as required.

RESET-OPTIONS

The selection of this feature is used to reset all DEBUG options to their default settings:

```
TRACE-ON

TRACE-FAST
```

Main Menu

EDIT

The EDIT function is used to enter your COBOL 64 statements. A set of EDIT functions are also provided to facilitate the editing process:

DIRECTORY	list the disk directory
LIST	list lines on screen
DELETE	delete lines
SYNTAX	syntax analysis
AUTO	auto line numbers
SAVE	save program on disk
RESEQUENCE	renumber all lines
PRINT-ON	set printer on
PRINT-OFF	set printer off
EXIT	exit to main menu

If the COEDIT program overlay is not present in memory it will automatically be loaded from disk by the system. Messages will appear on the screen when this occurs. When the START EDIT message appears on the screen you can begin entering COBOL 64 Statements or EDIT-Functions.

COBOL 64 Statements

Start by entering a six digit line number followed by the remainder of your statement. Refer to the section titled Editing Format for additional information. Each line must be terminated by the RETURN key before it is processed by the system. The Commodore cursor control keys are enabled including the insert/delete keys. Refer to your Commodore User's Guide for more details. One line on the screen is 40 characters, a COBOL 64 line can be up to 80 characters, which would occupy 2 lines on the screen. To replace a line you may simply type the new line with the same line number as the line your are replacing. The EDIT-LIST function described below can be used to

view your text. You can, for example, list a line or series of lines, and then type any changes needed followed by the RETURN key. Inserting lines is accomplished by typing a line number which falls between two existing lines. Deleting lines is accomplished by the EDIT-DELETE function described below.

While entering COBOL 64 statements some validation of the text is performed. If an error is detected INVALID ENTRY is displayed. If this message should appear, review the entry you have just typed and make any necessary corrections.

Example:

If you enter -

```
ADD A TO 1B
```

the INVALID ENTRY message will appear because 1B is not a valid COBOL 64 word.

Additional validation (Syntax Analysis) is performed at another time.

DIRECTORY

The DIRECTORY function will list the disk directory on the screen. The abbreviation DIR can be used.

LIST

The LIST function has the following format:

```
LIST [starting-line-number] [ending-line-number]
```

1. If no line numbers are present then the entire file is displayed.

2. If only one line number is entered then only that line is displayed (if present).

3. If two line numbers are entered (at least one space between the numbers is required) then the first number is interpreted as the starting line and the second as the ending line number. Note the ending line number must be greater than or equal to the starting line number. The lines are displayed if present.

4. The RUN/STOP key can be used to terminate the listing process or the listing process can be paused by holding down the SHIFT key.

5. The LIST function can be abbreviated 'L'.

6. Leading zeros on line numbers need not be entered.

7. An error message is displayed if invalid line numbers are entered; such as 12X3 or more than 6 digits.

DELETE

The format and validation of DELETE is similar to LIST. This function deletes the lines indicated. The lines are displayed for documentation purposes.

SYNTAX

This function performs a complete syntax analysis of your program. Any errors found will cause the line in error to be displayed (including the previous 7 lines) and an error message.

For the IDENTIFICATION, ENVIRONMENT and DATA divisions the syntax analysis process is aborted following the first error encountered. The PROCEDURE DIVISION is only syntaxed if no errors are found in the other divisions. All errors are reported in the PROCEDURE DIVISION.

This function is optional during the EDIT Mode. An automatic syntax analysis will be forced when you select the RUN or DEBUG Mode for any program. It has been made available in the EDIT Mode to allow you to selectively syntax portions of your program as you develop it.

The syntax analysis process involves two automatic program overlays to occur from disk, COSYN and COSYNP. Messages are displayed for this purpose. Following the syntax process the COEDIT program overlay is automatically reloaded along with appropriate messages appearing on the screen. This entire process (excluding optional printer time) will take less than two minutes, regardless of the program size.

AUTO

The AUTO function provides for automatic display of the next line number. The AUTO function has the following format:

```
AUTO [line-increment-value]
```

1. The line-number-value must be within the range 1 to 100.

2. If no value is entered then the AUTO function is turned off.

3. The line number displayed is computed by adding the line-increment-value to the last line entered into the system.

4. In addition to displaying the new line number, column 12 is indicated by displaying a large dot. This was provided to facilitate formatting the line. If no new text is entered in column 12, the large dot will be automatically removed by the system before validation begins. There is no need for you to type over it.

5. If the automatic increment should cause an overflow, an error message is displayed and the function is turned off.

SAVE

The SAVE function causes your program to be saved to disk, refer to Main Menu SAVE for additional details. It is good practice to periodically save your program file onto disk in case a problem with your computer or electrical power develops. It is also good practice to save your program on a second or third diskette in case a problem with the diskette develops.

RESEQUENCE

The RESEQUENCE function will renumber all lines in your program by increments of 100.

PRINT-ON PRINT-OFF

This function forces all keyboard input and displays to the screen to be printed on the printer. The printer must be powered on. The feature provides for creating program listings, documents all changes and error messages. For example, if the PRINT-ON is entered followed by LIST 5000 8000, these lines are displayed on the screen and printed on the printer. If SYNTAX is selected with the PRINT-ON, all error messages are printed for later evaluation.

Printing is terminated with the PRINT-OFF function.

These functions are also available from the COBOL 64 Main Menu selection.

EXIT

The EXIT function allows for terminating the EDIT session and returns processing to the COBOL 64 Main Menu. If the EDIT session included changing the program file, then a warning message appears, which will allow files to be saved on disk. Refer to the Main Menu SAVE function for additional details on saving a file.

Main Menu

GET

The GET function is used to get (load) an existing program file from disk into memory. A message appears on the screen requesting that a file name be entered. At this time you must remove the COBOL 64 diskette and insert your diskette. Enter the desired file name (enclosed in quotation marks) followed by the RETURN key. Messages will appear during the loading process. The system will return to the main menu following the loading process. At this time you may need to remove your diskette and insert the COBOL 64 diskette, depending on your next menu selection.

NEW-NAME

The NEW-NAME function provides for the changing of a program file name. A file must be present, refer to GET or NEW-PROG. A message appears on the screen requesting that the new file name be entered. Enter the new file name (enclosed in quotation marks) followed by the RETURN key. The system then returns to the Main Menu. This feature is useful when using an existing program as a basis for a new program.

NEW-PROG/EDIT

This function erases any existing program in memory and accepts a new program name. A message appears on the screen requesting that a file name be entered. Enter the new program name (enclosed in quotation marks) followed by the RETURN key. The system then proceeds as if the EDIT function had been selected from the Main Menu. Refer to the section on EDIT for additional information.

PRINT-ON PRINT-OFF

These features force all keyboard input and screen displays to be printed on the printer. The printer must be powered on. They are intended for creating program listings, documenting all changes, menu selections, tracing and error messages.

Printing is terminated with the PRINT-OFF function.

RUN

This function directs the system to begin executing the current program file in memory. A file must be present, refer to GET or NEW-PROG. Before execution can begin your program is tested to determine if it has been successfully syntaxed.

If it has, the CORD (RUN/DEBUG) program overlay is automatically loaded from disk if required. At the beginning of RUN Mode a "START RUN" message will appear on the screen. At this time you may need to remove the COBOL 64 diskette and insert your diskette if your program is going to use the disk drive. Enter any key to begin execution of your program. When your program execution is terminated, the system will return to the Main Menu.

If your program had not previously been successfully syntaxed, then the syntax process will begin. Refer to EDIT SYNTAX for additional details. If the syntax process is unsuccessful, the system resumes at the Main Menu. If the syntax process is successful, the system proceeds with the CORD program overlay as described above.

SAVE

The SAVE function provides for the saving of your program onto disk. A file must be present, refer to GET or NEW-PROG. Before proceeding with the SAVE, remove the COBOL 64 diskette and insert your diskette.

Program files are saved in two parts. Each part is prefixed by C1 or C2 before your file name. eg., C1YOURFILE C2YOURFILE. Screen messages will appear during the saving process. If your file already exists on the diskette it will automatically be overwritten by the new file.

APPENDIX A

SAMPLE PROGRAM

The following sample program is an example of one way to write a COBOL 64 program which performs the function of a simple adding machine. This program is available on the COBOL 64 diskette. It is recommended that you use the program to get familiar with the COBOL 64 system. After studying the program listing try the exercises below:

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. VS-ADDING-MACHINE.
000300 ENVIRONMENT DIVISION.
000400 CONFIGURATION SECTION.
000500 SOURCE-COMPUTER.  C64.
000600 OBJECT-COMPUTER. C64.
000700 DATA DIVISION.
000800 WORKING-STORAGE SECTION.
000900 01  DISPLAY-LINE.
001000       02 DISP-SPACE PIC X(20) VALUE " ".
001100       02 DISP-NUMBER PIC ZZ,ZZZ,ZZZ.99+.
001200 01  ENTRY.
001300       02 ENTRY-TABLE PIC X OCCURS 12 TIMES.
001400*    10 DIGITS + 1 DECIMAL POINT + 1 FUNCTION KEY
001500 77  ENTRY-SUB PIC 99.
001600 77  KEY-IN PIC X.
001700 77  TOTAL PIC S9(8)V99 VALUE 0.
001800 77  NUMERIC-ENTRY PIC S9(8)V99.
001900 77  CLEAR-HOME-CODE VALUE CHR 147 PIC X.
002000 77  RETURN-CODE VALUE CHR 13 PIC X.
002100 77  RVS-ON-CODE VALUE CHR 18 PIC X.
002200 77  RVS-OFF-CODE VALUE CHR 146 PIC X.
002300 PROCEDURE DIVISION.
002400 START.
002500     DISPLAY CLEAR-HOME-CODE
002600             "START ADDING MACHINE"
002700             RETURN-CODE.
```

```
002800 START-ENTRY.
002900     DISPLAY RETURN-CODE
003000             "ENTER: "
003100     MOVE " " TO ENTRY
003200     MOVE 1 TO ENTRY-SUB.
003300 ACCEPT-LOOP.
003400     ACCEPT-1-KEY KEY-IN
003500     DISPLAY KEY-IN
003600     IF KEY-IN IS EQUAL TO "+"
003700        PERFORM PLUS-KEY THRU PLUS-KEY-EXIT
003800        GO TO START-ENTRY.
003900     IF KEY-IN EQUAL "-"
004000        PERFORM MINUS-KEY THRU MINUS-KEY-EXIT
004100        GO TO START-ENTRY.
004200     IF KEY-IN EQUAL "S"
004300        PERFORM SUB-KEY
004400        GO TO START-ENTRY.
004500     IF KEY-IN EQUAL "T"
004600        PERFORM TOT-KEY
004700        GO TO START-ENTRY.
004800     IF KEY-IN EQUAL "E" STOP RUN.
004900*
005000     MOVE KEY-IN TO ENTRY-TABLE(ENTRY-SUB)
005100     ADD 1 TO ENTRY-SUB
005200     IF ENTRY-SUB IS GREATER THAN 12
005300         PERFORM ERR
005400         GO TO START-ENTRY
005500       ELSE GO TO ACCEPT-LOOP.
005600 PLUS-KEY.
005700     FILTER-NUMERIC ENTRY TO NUMERIC-ENTRY
005800         ON ERROR
005900            PERFORM ERR
006000            GO TO PLUS-KEY-EXIT.
006100     ADD NUMERIC-ENTRY TO TOTAL
006200     MOVE NUMERIC-ENTRY TO DISP-NUMBER
006300     DISPLAY RETURN-CODE
006400         DISPLAY-LINE.
006500 PLUS-KEY-EXIT. EXIT.
006600 MINUS-KEY.
```

```
006700     FILTER-NUMERIC ENTRY TO NUMERIC-ENTRY
006800         ON ERROR
006900            PERFORM ERR
007000            GO TO MINUS-KEY-EXIT.
007100     MULTIPLY -1 BY NUMERIC-ENTRY
007200     ADD NUMERIC-ENTRY TO TOTAL
007300     MOVE NUMERIC-ENTRY TO DISP-NUMBER
007400     DISPLAY RETURN-CODE
007500         DISPLAY-LINE.
007600 MINUS-KEY-EXIT. EXIT.
007700 SUB-KEY.
007800     IF ENTRY-SUB IS EQUAL TO 1
007900        MOVE TOTAL TO DISP-NUMBER
008000        DISPLAY RETURN-CODE
008100                DISPLAY-LINE
008200                KEY-IN
008300      ELSE PERFORM ERR.
008400 TOT-KEY.
008500     IF ENTRY-SUB EQUAL 1
008600        PERFORM SUB-KEY
008700        MOVE 0 TO TOTAL
008800      ELSE PERFORM ERR.
008900 ERR.
009000     DISPLAY RETURN-CODE
009100             RVS-ON-CODE
009200             "INVALID ENTRY"
009300             RVS-OFF-CODE.
009400 END-PROG.
```

APPENDIX A

Exercises

1. Load and execute the program in RUN Mode.

   a. From the Main Menu select GET file:

      Enter 5 then RETURN

   b. Enter the file name in quotes

      Enter "ADDING" then RETURN

At this point the ADDING program is loaded into memory.

   c. Select the RUN program option

      Enter 2 then RETURN

At this point the CORD program overlay will take place followed by screen messages "START RUN" and "ENTER ANY KEY TO CONTINUE".

   d. Enter any key to start the ADDING program.

   e. Enter some entries such as:

```
123+
456+
   S
   T
```

      Try some invalid entries such as:

```
12B3+     (not a number)
12.345+   (too many digits after decimal point)
```

   f. To exit the program and return to the Main Menu

      Enter E

2. Execute the program in DEBUG Mode.

   a. From the Main Menu and with the program already loaded into memory (Step 1 above).

      Enter 3 then RETURN

   Note that the CORD program overlay is not needed at this time because it is still in memory from Step 1 above.

   The DEBUG Menu is displayed at this time.

   b. Proceed at this point by selecting TRACE-SLOW.

      Enter 13 then RETURN.

   c. Select the START-PROG option.

      Enter 1 then RETURN

   The program will start executing with a display of each COBOL 64 statement as it is being executed. The trace display will stop when the ACCEPT-1-KEY statement is executed (line # 003400)

   d. Now enter some entries as you did in Step 1 above and try to follow the program execution sequence.

   e. Exit the program as you did in Step 1.

      Enter E

      The system returns to the DEBUG Menu.

   f. Return to the Main Menu by selecting EXIT.

      Enter 5 then RETURN

3. Try making some changes to the program so that you can exercise the EDIT Mode. Consider changing the size of the entry or even simpler, the "START ADDING MACHINE" on line 002600.

   a. From the Main Menu select EDIT Mode.

      Enter 1 then RETURN

   The COEDIT program overlay is then loaded into memory.

   b. List the program on the screen.

      Enter LIST then RETURN.

      Continue changing and listing as required.

   c. To exit the EDIT Mode

      Enter EXIT then RETURN

   Note the warning message to save the program. You cannot use the COBOL 64 diskette because it has the write protect tab on. Insert your own disk to save the program or bypass the save. If you do insert your disk be sure to replace it with the COBOL 64 diskette after the save has been completed. Now the Main Menu is present on the screen. Try executing your changes by following Step 1 or 2 above. This time you will observe that a Syntax Analysis is being performed, this will occur whenever a program is changed.

APPENDIX B

RESERVED WORDS

All reserved words known to the COBOL 64 System are listed in this Appendix.

ACCEPT	ACCEPT-1-KEY	ACCESS
ADD	ADVANCING	AFTER
ALPHABETIC	AT	AUTHOR
BEFORE	BY	CLOSE
COMMA	CONFIGURATION	CURRENCY
DATA	DATE-WRITTEN	DEBUG-BREAK
DEBUG-TRACE-OFF	DEBUG-TRACE-ON	DECIMAL-POINT
DEPENDING	DISPLAY	DIVIDE
DIVISION	END	ENVIRONMENT
EQUAL	ERROR	FD
FILE	FILE-CONTROL	FILLER
FILTER-NUMERIC	FROM	GIVING
GO	GREATER	I-O
IDENTIFICATION	IF	INDEX
IDEXED	INPUT	INPUT-OUTPUT
INSTALLATION	INTO	INVALID
IS	KEY	LABEL
LESS	LINE	LINES
MODE	MOVE	MULTIPLY
NEXT	NOT	NUMERIC
OBJECT-COMPUTER	OCCURS	OF
OMITTED	ON	OPEN
ORGANIZATION	OUTPUT	PERFORM
PIC	PICTURE	PROCEDURE
PROGRAM-ID	RANDOM	READ
RECORD	RECORDS	RELATIVE
ROUNDED	RUN	SECURITY
SELECT	SENTENCE	SEQUENTIAL
SET	SIGN	SIZE
SOURCE-COMPUTER	SPECIAL-NAMES	STANDARD
STATUS	STOP	SUBTRACT
THAN	THROUGH	THRU
TIMES	TO	VALUE
WORKING-STORAGE	WRITE	

APPENDIX C

LANGUAGE SUMMARY

```
IDENTIFICATION DIVISION
   PROGRAM-ID
   AUTHOR
   INSTALLATION
   DATE-WRITTEN
   SECURITY

ENVIRONMENT DIVISION
   CONFIGURATION SECTION
      SOURCE-COMPUTER
      OBJECT-COMPUTER
      SPECIAL-NAMES
         CURRENCY SIGN IS...
         DECIMAL-POINT IS COMMA
   INPUT-OUTPUT SECTION
   FILE-CONTROL
      SELECT...ASSIGN...
      ORGANIZATION IS SEQUENTIAL
         ACCESS MODE IS SEQUENTIAL
      ORGANIZATION IS RELATIVE
         ACCESS MODE IS SEQUENTIAL RELATIVE KEY IS ...
         ACCESS MODE IS RANDOM RELATIVE KEY IS...
      FILE STATUS IS...

DATA DIVISION
   FILE SECTION
      FD
         LABEL RECORDS ARE...
         VALUE OF FILE-ID IS...
   WORKING STORAGE SECTION
      LEVEL-NUMBER...FILLER...
         PICTURE IS...
         USAGE IS...INDEX...
         OCCURS...TIMES...
         INDEXED BY...
         VALUE IS...CHR...
```

LANGUAGE SUMMARY continued

```
PROCEDURE DIVISION
   ACCEPT...
   ACCEPT-1-KEY...
   ADD...GIVING...ROUNDED ON SIZE ERROR...
   CLOSE...
   DEBUG-BREAK
   DEBUG-TRACE-OFF
   DEBUG-TRACE-ON
   DISPLAY...
   DIVIDE...INTO BY...GIVING...ROUNDED ON SIZE ERROR...
   EXIT
   FILTER-NUMERIC...
   GO TO...DEPENDING ON...
   IF...NEXT SENTENCE...ELSE...NEXT SENTENCE...
   MOVE...
   MULTIPLY...BY...GIVING...ROUNDED ON SIZE ERROR...
   OPEN INPUT...OUTPUT...I-O...
   PERFORM...THRU...
   READ...AT END...INVALID KEY...
   SET...
   STOP RUN
   SUBTRACT...FROM...GIVING...ROUNDED ON SIZE ERROR...
   WRITE...BEFORE/AFTER ADVANCING...LINES...INVALID KEY...
```

APPENDIX D

SAMPLE PROGRAMS

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.  BUILD-DATA1.
000300 AUTHOR. K A  ALEXANDER.
000400 ENVIRONMENT DIVISION.
000500 CONFIGURATION SECTION.
000600 SOURCE-COMPUTER.  C64.
000700 OBJECT-COMPUTER.  C64.
000800 INPUT-OUTPUT SECTION.
000900 FILE-CONTROL.
001000     SELECT DATA1 ASSIGN TO DISK-1541 DRIVE-8
001100     FILE STATUS IS FILE-ST.
001200 DATA DIVISION.
001300 FILE SECTION.
001400 FD  DATA1
001500     LABEL RECORDS ARE OMITTED
001600     VALUE OF FILE-ID IS "00:DATA1".
001700 01  DATA-RECORD.
001800     02  NAME-FIELD   PIC X(20).
001900     02  ADDR-FIELD   PIC X(20).
002000 01  DATA-RECORD2.
002100     02  NAME-FIELD-EXIT PIC X(4).
002200     02  FILLER PIC X(36).
002300 WORKING-STORAGE SECTION.
002400 77  WRITE-FLAG PIC X VALUE "N".
002500 77  RVS-ON VALUE CHR 18 PIC X.
002600 77  RETURN-CODE VALUE CHR 13 PIC X.
002700 77  CLEAR-HOME VALUE CHR 147 PIC X.
002800 77  FILE-ST PIC XX.
002900 PROCEDURE DIVISION.
003000 START-UP.
003100     DISPLAY CLEAR-HOME
003200     OPEN OUTPUT DATA1
003300     IF FILE-ST IS NOT EQUAL TO
003400        "00" DISPLAY "OPEN ERROR"
003500             STOP RUN.
003600     PERFORM GET-DATA-LOOP THRU LOOP-EXIT.
003700 END-IT.
```

APPENDIX D

SAMPLE PROGRAMS

```
003800     CLOSE DATA1
003900     IF FILE-ST NOT EQUAL TO "00"
004000        DISPLAY "CLOSE ERROR".
004100     STOP RUN.
004200 GET-DATA-LOOP.
004300     DISPLAY RVS-ON
004400       "ENTER NAME FIELD                "
004500        RETURN-CODE
004600     ACCEPT NAME-FIELD
004700     IF NAME-FIELD IS NOT ALPHABETIC
004800        DISPLAY "NOT ALPHA"
004900                RETURN-CODE
005000        GO TO GET-DATA-LOOP.
005100     IF NAME-FIELD-EXIT EQUAL TO
005200       "EXIT" GO TO LOOP-EXIT.
005300     DISPLAY RVS-ON
005400       "ENTER ADDRESS                   "
005500       RETURN-CODE
005600     ACCEPT ADDR-FIELD
005700     DISPLAY "DATA OK? (Y/N)".
005800     ACCEPT WRITE-FLAG.
005900     IF WRITE-FLAG EQUAL "Y"
006000        PERFORM WRITE-ROUTINE.
006100     GO TO GET-DATA-LOOP.
006200 WRITE-ROUTINE.
006300     WRITE DATA-RECORD.
006400     IF FILE-ST NOT EQUAL TO "00"
006500        DISPLAY "WRITE ERROR"
006600        STOP RUN.
006700     MOVE " " TO DATA-RECORD.
006800     MOVE "N" TO WRITE-FLAG.
006900 LOOP-EXIT.
007000     EXIT.
**COMPLETED
PRINT-OFF
```

APPENDIX D

SAMPLE PROGRAMS

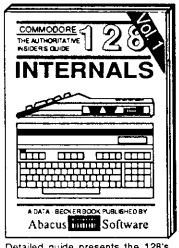
```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.  LIST-DATA1.
000300 AUTHOR. K A ALEXANDER.
000400 ENVIRONMENT DIVISION.
000500 CONFIGURATION SECTION.
000600 SOURCE-COMPUTER.  C64.
000700 OBJECT-COMPUTER.  C64.
000800 INPUT-OUTPUT SECTION.
000900 FILE-CONTROL.
001000     SELECT DATA1 ASSIGN TO DISK-1541 DRIVE-8
001100     FILE STATUS IS FILE-ST.
001200     SELECT PRINT-FILE ASSIGN TO PRINTER-1525.
001300 DATA DIVISION.
001400 FILE SECTION.
001500 FD  DATA1
001600     LABEL RECORDS ARE OMITTED
001700     VALUE OF FILE-ID IS "DATA1".
001800 01  DATA1-RECORD PIC X(40).
001900 FD  PRINT-FILE
002000     LABEL RECORDS ARE OMITTED.
002100 01  PRINT-REC PIC X(40).
002200 WORKING-STORAGE SECTION.
002300 77  FILE-ST PIC XX.
002400 PROCEDURE DIVISION.
002500 START-UP.
002600     OPEN INPUT DATA1.
002700     IF FILE-ST NOT EQUAL TO "00"
002800        DISPLAY "OPEN ERROR"
002900        STOP RUN.
003000     OPEN OUTPUT PRINT-FILE.
003100     PERFORM READ-WRITE-LOOP THRU LOOP-EXIT.
003200 END-UP.
003300     CLOSE PRINT-FILE.
003400     CLOSE DATA1.
003500     IF FILE-ST NOT EQUAL TO "00"
003600        DISPLAY "CLOSE ERROR".
003700     STOP RUN.
```

APPENDIX D

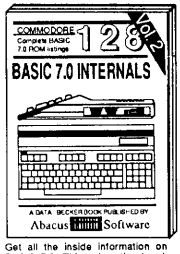
SAMPLE PROGRAMS

```
003800 READ-WRITE-LOOP.
003900     READ DATA1 AT END GO TO LOOP-EXIT.
004000     IF FILE-ST NOT EQUAL TO "00"
004100        DISPLAY "READ ERROR"
004200        STOP RUN.
004300     MOVE DATA1-RECORD TO PRINT-REC.
004400     WRITE PRINT-REC.
004500     MOVE " " TO PRINT-REC.
004600     GO TO READ-WRITE-LOOP.
004700 LOOP-EXIT.
004800     EXIT.
**COMPLETED
PRINT-OFF
```

C-128™ REQUIRED and C-64™ READING



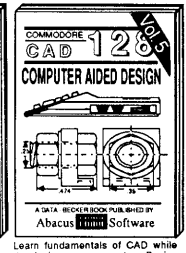
Detailed guide presents the 128's operating system, explains graphic chips, Memory Management Unit, 80 column graphics and commented ROM listings. **500pp $19.95**



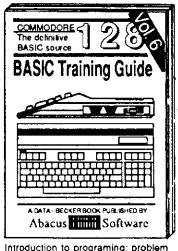
Get all the inside information on BASIC 7.0. This exhaustive handbook is complete with commented BASIC 7.0 ROM listings. Coming Summer'86. **$19.95**

Filled with info for everyone. Covers 80 column hi-res graphics, windowing, memory layout, Kernal routines, sprites, software protection, autostarting. **300pp $19.95**

Insiders' guide for novice & advanced users. Covers sequential & relative files, & direct access commands. Describes DOS routines. Commented listings. **$19.95**

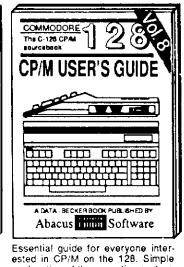


Learn fundamentals of CAD while developing your own system. Design objects on your screen to dump to a printer. Includes listings for '64 with Simon's Basic. **300pp $19.95**

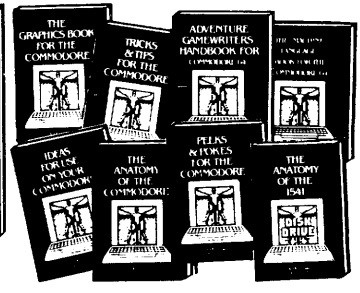


Introduction to programing; problem analysis, thorough description of all BASIC commands with hundreds of examples; monitor commands; utilities, much more. **$16.95**

Presents dozens of programming quick-hitters. Easy and useful techniques on the operating system, stacks, zero-page, pointers, the BASIC interpreter and more. **$16.95**



Essential guide for everyone interested in CP/M on the 128. Simple explanation of the operating system, memory usage, CP/M utility programs, submit files & more. **$19.95**



ANATOMY OF C-64 Insider's guide to the '64 internals. Graphics, sound, I/O, kernal, memory maps, more. Complete commented ROM listings. **300pp $19.95**

ANATOMY OF 1541 DRIVE Best handbook on [illegible] all. Many examples and [illegible] commented 1541 ROM listings. **500pp $19.95**

MACHINE LANGUAGE C-64 Learn 6510 code write fast programs. Many samples and listings for complete assembler, monitor, & simulator. **200pp $14.95**

GRAPHICS BOOK C-64 - best reference covers basic and advanced graphics. Sprites, animation, Hires, Multicolor, lightpen, 3D-graphics, IRQ, CAD, projections, curves, more. **350pp $19.95**

TRICKS & TIPS FOR C-64 Collection of easy-to-use techniques: advanced graphics, improved data input, enhanced BASIC, CP/M, more. **275pp $19.95**

1541 REPAIR & MAINTENANCE Handbook describes the disk drive hardware. Includes schematics and techniques to keep 1541 running. **200pp $19.95**

ADVANCED MACHINE LANGUAGE Not covered elsewhere: - video controller, interrupts, timers, clocks, I/O, real time, extended BASIC, more. **210pp $14.95**

PRINTER BOOK C-64/VIC-20 Understand Commodore, Epson-compatible printers and 1520 plotter. Packed; utilities; graphics dump; 3D-plot; commented MPS801 ROM listings, more. **330pp $19.95**

SCIENCE/ENGINEERING ON C-64 In depth intro to computers in science. Topics: chemistry, physics, biology, astronomy, electronics, others. **350pp $19.95**

CASSETTE BOOK C-64/VIC-20 Comprehensive guide; many sample programs. High speed operating system fast file loading and saving. **225pp $14.95**

IDEAS FOR USE ON C-64 Themes: auto expenses, calculator, recipe file, stock lists, diet planner, window advertising, others. Includes listings. **200pp $12.95**

COMPILER BOOK C-64/C-128 All you need to know about compilers: how they work; designing and writing your own; generating machine code. With working example compiler. **300pp $19.95**

Adventure Gamewriter's Handbook Step-by-step guide to designing and writing your own adventure games. With automated adventure game generator. **200pp $14.95**

PEEKS & POKES FOR THE C-64 Includes in-depth explanations of PEEK, POKE, USR, and other BASIC commands. Learn the "inside" tricks to get the most out of your '64. **200pp $14.95**

Optional Diskettes for books For your convenience, the programs contained in each of our books are available on diskette to save you time entering them from your keyboard. Specify name of book when ordering. **$14.95 each**

C-128 and C-64 are trademarks of Commodore Business Machines Inc.

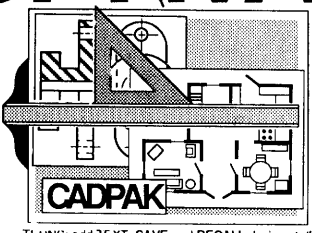
Abacus Software

P.O. Box 7219 Grand Rapids, MI 49510 - Telex 709-101 - Phone (616) 241-5510

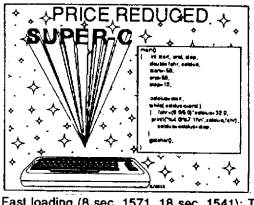
Call **now** for the name of your nearest dealer. Or to order directly by credit card, MC, AMEX of VISA call **(616) 241-5510**. Other software and books are available–Call and ask for your **free** catalog. Add $4.00 for shipping per order. Foreign orders add $10.00 per book. Dealer inquires welcome–1400+ nationwide.

'128™ and C-64™ SPECTACULAR SOFTWARE

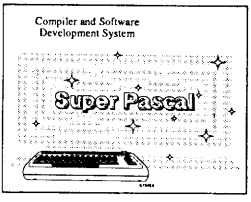
The *complete* compiler and development package. Speed up your programs 5x to 35x. Many options: flexible memory management; choice of compiling to machine code, compact p-code or both. '128 version: 40 or 80 column monitor output and FAST-mode operation. '128 Compiler's extensive 80-page programmer's guide covers compiler directives and options, two levels of optimization, memory usage, I/O handling, 80 column hi-res graphics, faster, higher precision math functions, speed and space saving tips, more. A great package that no software library should be without. **128 Compiler $59.95**
64 Compiler $39.95



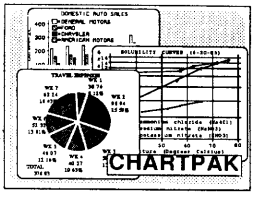
Remarkably easy-to-use interactive drawing package for accurate graphic designs. New dimensioning features to create exact scaled output to all major dot-matrix printers. Enhanced version allows you to input via keyboard or high quality lightpen. Two graphic screens for COPYing from one to the other. DRAW, LINE, BOX, CIRCLE, ARC, ELLIPSE available. FILL objects with preselected PATTERNS; add TEXT; SAVE and RECALL designs to/from disk. Define your own library of symbols/objects with the easy-to-use OBJECT MANAGEMENT SYSTEM–store up to 104 separate objects. **C-128 $59.95**
C-64 $39.95



For school or software development. Learn C on your Commodore with our in-depth tutorial. Compile C programs into *fast* machine language. C-128 version has added features: Unix™-like operating system; 60K RAM disk for fast editing and compiling Linker combines up to 10 modules; Combine M/L and C using CALL; 51K available for object code; Fast loading (8 sec. 1571, 18 sec. 1541); Two standard I/O librarys plus two additional libraries—math functions (sin, cos, sqrt, etc.) & 20+ graphic commands (line, fill, dot, etc.). **C-128 $79.95**
C-64 $79.95



Not just a compiler, but a complete system for developing applications in Pascal with graphics and sound features. Extensive editor with search, replace, auto, renumber, etc. Standard J & W compiler that generates fast machine code. If you want to learn Pascal or to develop software using the best tools available–**SUPER Pascal** is your first choice. **C-128 $59.95**
C-64 $59.95



Easily create professional high quality charts and graphs without programming. You can immediately change the scaling, labeling, axis, bar- filling, etc. to suit your needs. Accepts data from CalcResult and MultiPlan. C-128 version has 3X the resolution of the '64 version. Outputs to most printers. **C-128 $39.95**
C-64 $39.95

OTHER TITLES AVAILABLE:

Technical Analysis System

Sophisticated charting and technical analysis system for serious investors. Charting and analyzing past history of a stock, TAS can help pinpoint trends & patterns and predict a stock's future. Enter data from the keyboard or from online financial services. **C-64 $59.95**

Personal Portfolio Manager

Complete protfolio management system for the individual or professional investor. Easily manage your portfolios, obtain up-to-the-minute quotes and news, and perform selected analysis. Enter quotes manually or automatically through Warner Computer Systems. **C-64 $39.95**

Xper

XPER is the first "*expert system*" for the C-128 and C-64. While ordinary data base systems are good for reproducing facts, XPER can derive knowledge from a mountain of facts and help you make expert decisions. Large capacity. Complete with editing and reporting. **C-64 $59.95**

PowerPlan

One of the most powerful spreadsheets with integraded graphics. Includes menu or keyword selections, online help screens, field protection, windowing,trig functions and more. PowerGraph, the graphics package, is included to create integrated graphs & charts. **C-64 $39.95**

COBOL Compiler for the C-64	**$39.95**
Ada Compiler for the C-64	**$39.95**
VideoBasic Language for the C-64	**$39.95**

Abacus Software

P.O. Box 7219 Grand Rapids, MI 49510 - Telex 709-101 - Phone (616) 241-5510

Call **now** for the name of your nearest dealer. Or to order directly by credit card, MC, AMEX of VISA call **(616) 241-5510**. Other software and books are available–Call and ask for your **free** catalog. Add $4.00 for shipping per order. Foreign orders add $12.00 per item. Dealer inquires welcome–1400+ nationwide.

ATARI® ST™

REQUIRED READING

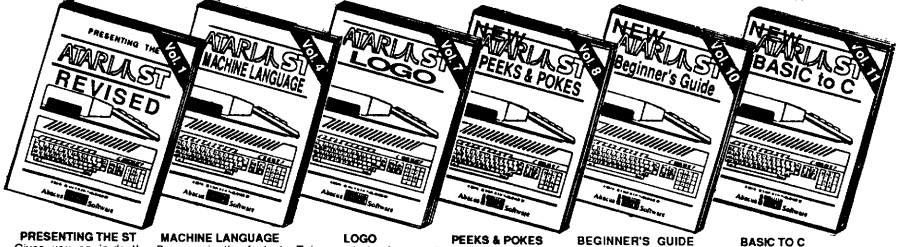
INTERNALS
Essential guide to learning the inside information of the ST. Detailed descriptions of sound & graphics chips, internal hardware, various ports, GEM. Commented BIOS listing. An indispensible reference for your library. 450pp. $19.95

GEM Programmer's Ref.
For serious programmers in need of detailed information on GEM. Written with an easy-to-understand format. All GEM examples are written in C and assembly. Required reading for the serious programmer. 450pp. $19.95

TRICKS & TIPS
Fantastic collection of programs and info for the ST. Complete programs include: super-fast RAM disk; time-saving printer spooler; color print hardcopy; plotter output hardcopy. Money saving tricks and tips. 200 pp. $19.95

GRAPHICS & SOUND
Detailed guide to understanding graphics & sound on the ST. 2D & 3D function plotters, Moiré patterns, various resolutions and graphic memory, fractals, waveform generation. Examples written in C, LOGO, BASIC and Modula2. $19.95

BASIC Training Guide
Indispensible handbook for beginning BASIC programmers. Learn fundamentals of programming. Flowcharting, numbering system, logical operators, program structures, bits & bytes, disk use, chapter quizzes. 200pp. $16.95



PRESENTING THE ST
Gives you an in-depth look at this sensational new computer. Discusses the architecture of the ST, working with GEM, the mouse, operating system, all the various interfaces, the 68000 chip and its instructions, LOGO. $16.95

MACHINE LANGUAGE
Program in the fastest language for your Atari ST. Learn the 68000 assembly language, its numbering system, use of registers, the structure & important details of the instruction set, and use of the internal system routines. 280pp $19.95

LOGO
Take control of your ATARI ST by learning LOGO–the easy-to-use, yet powerful language. Topics covered include structured programming, graphic movement, file handling and more. An excellent book for kids as well as adults. $19.95

PEEKS & POKES
Enhance your programs with the examples found within this book. Explores using the different languages BASIC, C, LOGO and machine language, using various interfaces, memory usage, reading and saving from and to disk, more. $16.95

BEGINNER'S GUIDE
Finally a book for those new to the ST wanting to understanding ST basics. Thoroughly understand your ST and its many devices. Learn the fundamentals of BASIC, LOGO and more. Complete with index, glossary and illustrations. +200pp $14.95

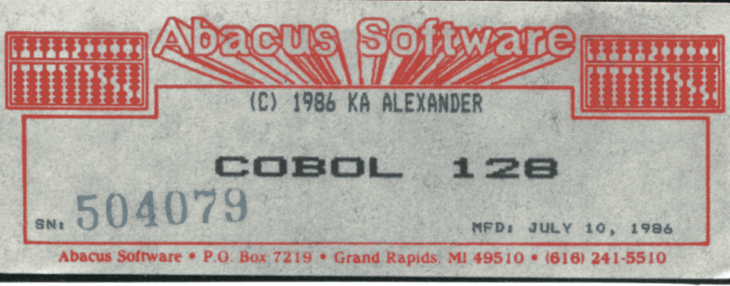
BASIC TO C
If you are already familiar with BASIC, learning C will be all that much easier. Shows the transition from a BASIC program, translated step by step, to the final C program. For all users interested in taking the next step. $19.95

Abacus Software

P.O. Box 7219 Grand Rapids, MI 49510 - Telex 709-101 - Phone (616) 241-5510

Optional diskettes are available for all book titles at **$14.95**

Call **now** for the name of your nearest dealer. Or order directly from ABACUS with your MasterCard, VISA, or Amex card. Add $4.00 per order for postage and handling. Foreign add $10.00 per book. **Other software and books coming soon.** Call or write for your **free** catalog. Dealer inquiries welcome–over 1400 dealers nationwide.


Abacus Software
(C) 1986 KA ALEXANDER
COBOL 128
SN: 504079
MFD: JULY 10, 1986
Abacus Software • P.O. Box 7219 • Grand Rapids MI 49510 • (616) 241-5510

You Can Count On
Abacus Software
P.O. Box 7219
Grand Rapids, MI 49510

REGISTRATION CARD

Registration # 504079 Product:_____

Name_____

Address_____

_____

City_____ State_____ Zip_____

Purchase Information:

Dealer_____

Address_____

City_____ State_____ Zip_____

Returning this registration card entitles you to phone support for the above product. You may also obtain a backup copy of the diskette for a handling charge of $10.00. This card and a check, money order or credit card number must accompany this request. Purchase orders are not acceptable.

BACKUP COPY? ____ No, do not send a backup, but register my purchase

____ Yes, send a backup copy, payment is enclosed

Credit card#_____

Expiration Date____/____/____

SEND NO MONEY

Return this Certificate for a FREE ISSUE of RUN Magazine!

Complete this certificate and mail today to see for yourself, the money-saving, time-saving help you'll get in every issue of *RUN*—the Commodore C-128/C-64 Home Computing Guide! Act now and discover the many ways *RUN* can increase the value of your computer every month with:

- **Programs & Utilities** that give you a wider range of applications...smoother running programs than ever before.
- **Expert Reviews** of hardware and software for your C-128 or C-64 that will steer you straight...help you save money...avoid costly mistakes.
- **Special Departments** on telecommunications, education, News from Commodore, solving problems, and more!

YES, send my **FREE EXAMINATION ISSUE** of *RUN* and enter my no-risk Subscription for one year (11 more monthly issues), for just **$19.97**—a savings of $15.43 **(44%)** off the newsstand price. If I decide not to subscribe, I will mark your invoice "cancel" and owe nothing. The FREE issue is mine to keep.

(print) Name

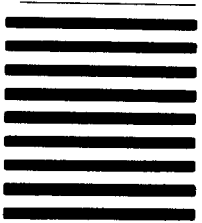
Address

City

State Zip

166PAA

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED
STATES



BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 73 PETERBOROUGH, NH

POSTAGE WILL BE PAID BY ADDRESSEE

CW Communications/Peterborough
RUN
P.O. Box 954
Farmingdale, NY 11737