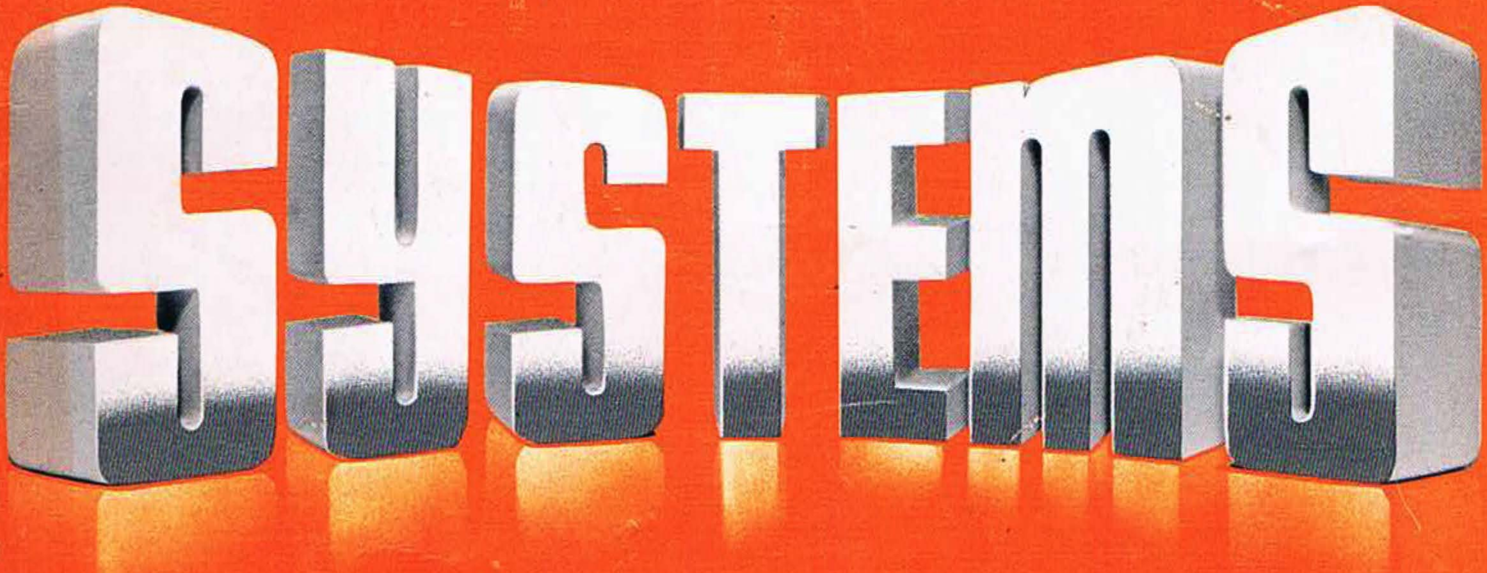
MOTOROLA

M68COBO(D)

M6800 RESIDENT COBOL OPERATIONS REFERENCE MANUAL



MICROSYSTEMS

M6800 RESIDENT COBOL OPERATIONS REFERENCE MANUAL COBOL 1.0

This manual describes the use and operation of the Motorola Resident COBOL compiler. An associated manual—RESIDENT COBOL LANGUAGE REFERENCE MANUAL—describes the programming language features of the compiler.

Second Edition
Copyright 1978 by Motorola Inc.
First Edition Feb., 1978

FOREWORD

This manual describes the use and operation of the Motorola M6800 COBOL compiler and COBOL operating environment. It assumes a general knowledge of the Motorola Disk Operating System (MDOS) as well as the operation of the EXORterm, EXORdisk, and EXORprint hardware. The following manuals should be referred to for additional information.

- M6800 Resident COBOL Language Reference Manual—M68COB(D)
- Motorola Disk Operating System (MDOS) User's Guide
- EXORterm User's Guide—M68SXS(D)
- EXORdisk II Floppy Disk Controller Module User's Guide M68SFDC(D)
- EXORprint Printer Manual
- EXORsystem User Manual

PREFACE

M6800 ANS COBOL is based on the specification of the COBOL standard published by the American National Standards Institute (formerly known as the United States of American Standards Institute) and contained in the publication USA Standard COBOL X3.23—1974.

As its name implies, COBOL (COmmon Business Oriented Language) is especially efficient in the processing of business problems. Such problems typically involve relatively little algebraic or logical processing; instead, they most often manipulate large files of basically similar records in a relatively simple way. This means that COBOL emphasizes mainly the description and handling of data items and input/output records.

This publication explains the use and operation of Motorola M6800 ANS COBOL and also includes information on the operating system. For information on the COBOL language syntax and features of COBOL that are supported by M6800 COBOL refer to the associated manual—MOTOROLA M6800 COBOL—LANGUAGE REFERENCE MANUAL.

ACKNOWLEDGEMENT

In compliance with the request of the Executive Committee of the Conference on Data Systems Languages (CODASYL), and specifically the CODASYL COBOL Committee, the following acknowledgement is extracted from that contained in the publication *COBOL, Edition 1974.*†

"Any organization interested in reproducing the COBOL report and specifications, in whole or in part, using ideas taken from this report as the basis for an instruction manual or for any other purpose is free to do so. However, all such organizations are requested to reproduce this section as part of the introduction to the document. Those using a short passage, as in a book review, are requested to mention COBOL in acknowledgement of the source, but need not quote this entire section.

"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 COBOL 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.

"Procedures have been established for the maintenance of COBOL. Inquiries concerning the procedures for proposing changes should be directed to the Executive Committee of the Conference on Data Systems Languages (CODASYL).

"The authors and copyright holders of the copyrighted material used herein 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."

---

†*COBOL, Edition 1974,* produced by joint efforts of the CODASYL COBOL Committee and the European Computer Manufacturers Association (ECMA).

CONTENTS

CHAPTER 1
M6800 COBOL

1.1 INTRODUCTION

Motorola's Resident ANS COBOL can be used with either EXORterm or EXORsystems equipment. EXORterms are a family of products primarily used in an engineering or systems equipment development environment. EXORsystems are a family of products oriented towards software development and applications use in the end-use applications environment. In either case, the use of Motorola's Resident ANS COBOL is virtually the same.

The major skill required to do an effective job of programming under any set of conditions is to have the ability to, first, clearly set down the results to be produced, and second, evolve a complete process of logic to achieve these results. Thereafter, programming becomes translating logic into some machine-acceptable language.

COBOL is such a language utilizing simple English-type statements which conform with the ANS standards. These statements and the nature of their use in a complete COBOL program appear in a separate Motorola Microsystems Publication entitled "M6800 Resident COBOL Language Reference Manual" (M68COB). The subject of this manual is not the COBOL language itself, but is the operation and use of COBOL on Motorola's microprocessor equipment.

1.1.1 Hardware Requirements

A minimum equipment configuration on which Motorola ANS COBOL programs can be compiled, edited, and finally run is:

2.1 Minimum EXORterm

- EXORterm 200 (or equivalent)
- 32K Bytes of Memory
- EXORdisk II
- EXORterm COBOL

In addition, the optional EXORprint is recommended.

2.2 Minimum EXORsystem

EXORsystem 500

1.1.2 Software Utilities

All software utilities required to compile, run, and edit programs are incorporated in one form or another into either of the minimum configurations indicated above. The user need not be concerned about how to get these utilities or where they are; only their use. For information purposes, however, these utilities include:

Disk Operating System (MDOS or MODOS)
COBOL Compiler and Run Time Library
COBOL Editor
File Management System

The creation and maintenance of the COBOL source programs are accomplished by using the editor. The source programs are then compiled by the COBOL compiler to produce the program load module. The editor and compiler as well as a number of utility programs operate under the direction of the disk operating system—MDOS. Prior to attempting the writing or compilation of COBOL programs with this system, the programmer should have some general knowledge of the use of the MDOS operating system and should be aware of the various MDOS utility programs available as an aid to software development. Information on the MDOS operating system may be found in the Motorola Disk Operating System—User's Manual.

1.2 ENTERING A NEW COBOL PROGRAM

The various chapters of this reference manual explain in detail how to write, edit, compile, and maintain COBOL programs. This section gives a brief overview of the entire process. In the following examples, all computer dialogue will be shown in upper case. *All system commands and COBOL source statements that would be typed by the programmer are underlined.* All user input is terminated by a carriage return.

1.2.1 Readying the system

The heart of both the EXORterm and the EXORsystem configurations is the CRT terminal in whose housing the actual microcomputer, memory, interfaces and other elements are incorporated. Considerable flexibility is designed into the CRT to accommodate a variety of different environments and configurations. The three groups of option switches on the back of the terminal housing itself must be set as follows for proper operation of the system:

GROUP ONE		GROUP TWO	
ENABLE	ON	DUPLEX	FULL
DISPLAY	OFF	PARITY	NO
TRANS MODE	OFF		EVEN
VIDEO INV	OFF	XMIT WORDS	8-BIT
A	ON	STOP BIT	1
B	OFF	CONNECTION	DIRECT
C	OFF	MODEM TYPE	103
SPEC CHAR	OFF	TURN AROUND	8-CHAN
LINE FREQ	OFF	CODE SEL	EOT

GROUP THREE
BAUD RATE—SELECT ONLY ONE. 9600 is the most desirable.

1.2.2 Loading the disk operating system (MDOS)

The actual procedure for initially loading the MDOS disk operating system will vary slightly between different hardware configurations. Refer to the MDOS user's guide for the exact initializing procedure for your hardware configuration. Typical procedures are:

EXORterm

```
depress restart key
EXBUG 1.2 MAID (no carriage return)
*E800;G (no carriage return)
MDOS X.X (X.X is version number)
=
```

EXORsystem

```
turn power on
EXORSYSTEM 500
=
```

The equal sign is an MDOS prompt character. User commands may now be entered.

1.2.3 Entering a new program

To enter a new COBOL program, the editor "BUILD" command is used. This command will create a COBOL source program file using the name given on the "BUILD" command (In this case: SAMPLE). The editor will prompt for each line by typing a new line number. The prompt/input sequence is terminated by depressing only the carriage return.

```
=EDIT
MOTOROLA EDITOR
? BUILD SAMPLE
0010  IDENTIFICATION DIVISION.
0020  PROGRAM-D. SAMPLE.
0030  ENVIRONMENT DIVISION.
0040  SOURCE COMPUTER. M6800.
0050  OBJECT COMPUTER. M6800.
0060  DATA DIVISION.
0070  WORKING-STORAGE SECTION.
0080  77 HELLO PIC X(5) VALUE 'HELLO'.
0090  PROCEDURE DIVISION.
0100  DISPLAT HELLO.
0110  carriage return
? SAVE
READY
?END
=
```

The sample COBOL program has now been saved on disk.

1.2.4 Compiling the program

In this case, we want to compile the program "SAMPLE" and put the generated load module in the file "FKEY01." No listing is to be produced.

```
=COBOL SAMPLE;0=FKEY01
MOTOROLA COBOL
0100  DISPLAT HELLO.
*ERROR 05  *
ERRORS IN PROGRAM
=
```

In this example, a syntax error is detected because the word DISPLAY is misspelled.

1.2.5 Editing the program

We now want to edit the program and re-type the line containing the error.

```
=EDIT
MOTOROLA EDITOR
?LOAD SAMPLE
READY
?LIST 100
0100  DISPLAT HELLO.
?100  DISPLAY HELLO.
?SAVE
READY
?END
=
```

The program is now recompiled. No errors are detected.

```
=COBOL SAMPLE;0=FKEY01
MOTOROLA COBOL
=
```

1.2.6 Execution

The source program "SAMPLE" has been compiled and given the load module name "FKEY01." This program may be executed by loading the COBOL operating system and depressing function key number one.

```
=RUN
READY—depress function key one
HELLO
READY—STOP
=
```

The system is now back in MDOS mode awaiting new MDOS commands as EDIT and COBOL.

The next chapters will examine the above process in greater detail.

1.3 EDITING A COBOL PROGRAM

COBOL programs are created and maintained by using the system editor. The editor is line-oriented. All program files have a four-digit line number followed by a blank at the start of each line. The editor is loaded by typing the MDOS command "EDIT."

Example:

```
=EDIT
```

There are two levels of edit features available: a basic set, which the user may master in a relatively short period of time, and an advanced set, which gives the user much greater flexibility in editing. The advanced set may be mastered as needed.

The basic command set includes the BUILD, LOAD, LIST, SAVE and END commands along with the elementary edit feature.

The EDITOR prompts the user for the next command by displaying a question mark. The command is entered by typing a carriage return. Many of the commands have an abbreviated form, or alias, that the programmer may use if desired. A description of the EDITOR commands follows.

1.3.1 Entering a new program

To enter a new program, use the edit command "BUILD."

BUILD—to create a new file in the work space

```
SYNTAX:     BUILD FFF.SS:L
            BUILD FFF.SS
            BUILD FFF
ALIAS:      BLD,NEW, CREATE
WHERE:      FFF is the file being created (8-character max)
            SS is the suffix (default = SA)
            L is the logical drive No. (default=0)
```

Examples:

```
BUILD PRTALL
BLD PRTALL.SA
NEW PRTALL.SA:0
```

The editor will now start prompting the user for input records.

The editor will prefix each line with a line number. The line numbers will start at 0010 and increment by ten for each new line. The input mode may be terminated by typing only a carriage return on the line. See the number command for additional information.

1.3.2 Editing an Existing Program

LOAD—use "LOAD" to load an already existing program into the work space.

```
SYNTAX:     LOAD FFF.SS:L
            LOAD FFF.SS
            LOAD FFFF
ALIAS:      LD, OLD, EDIT
WHERE:      FFF is the file name (8-character max)
```

	SS is the file suffix (default—SA)
	L is the logical drive No. (default = 0)
NOTE:	If specified, the suffix and logical drive become new default values for these parameters.

Examples:

```
LOAD PRTALL
LD PRTALL.SA
OLD PRTALL.SA:0
```

The following editor commands may be used to edit the program in the work space:

LIST—use "LIST" to display a line (or group of lines) on the console.

SYNTAX:	LIST
	LIST NN
	LIST NN-MM
ALIAS:	L
DEFAULT:	entire work space or block
WHERE:	NN is the first line number
	MM is the last line number
NOTE:	L 9999 will return with the last line number and amount of available memory remaining.
NOTE:	The "PRINT" command functions the same as "LIST" but the output is directed to the printer.

Example:

```
?LIST 10-30
0010  IF SALES EQUAL ZERO THEN PERFORM X-10.
0020  GO TO X20.
0030  STOP RUN.
```

SAVE—use "SAVE" to write the current program back to the disk.

SYNTAX:	SAVE
	SAVE FFFF.SS:L
	SAVE FFFF
ALIAS:	SAV
WHERE:	FFFF is the file name
	SS is the file suffix
	L is the logical drive No.
NOTE:	If a file name suffix or drive is not specified the data from the last "LOAD" or "BUILD" command is used again.
NOTE:	The disk file is not modified until the save command is executed.

Example:

```
?SAVE
READY
?
```

END—use "END" to leave the editor.

SYNTAX:	END
ALIAS:	QUIT,Q,EXIT
NOTE:	If the current program has been modified and not saved the "SAVE (Y/N)?" prompt will appear as a reminder of the currently unsaved status. A "Y" response will cause the current program to be written back to the last file specified. An "N" response will leave the program as it was on the disk.

Example:

```
?END
=
```

Elementary editing has three editing modes, insert, modify, and delete.

INSERT—to add a new line to the program in the work space

SYNTAX:	NNNN -----STRING-----
WHERE:	NNNN is a four-digit maximum line number -----STRING-----is the content of the new record.
NOTE:	A blank should immediately follow the line number.

Example:

```
?LIST 10-30
0010  MOVE 10 to AA.
0020  ADD 1 to BB.
0030  PERFORM X10.
?22 SUBTRACT 1 FROM CC.
?LIST 10-30.
0010  MOVE 10 TO AA.
0020  ADD 1 TO BB.
0022  SUBTRACT 1 FROM CC.
0030  PERFORM X10.
```

MODIFY—to change a line in the program in the work space

SYNTAX: (same as insert but uses an existing line number)

Example:

```
?LIST10-30
0010  MOVE 10 TO AA.
0020  ADD 1 TO BB.
0030  PERFORM X10.
?20 ADD 2 TO BB.
?LIST 10-30
0010  MOVE 10 TO AA.
0020  ADD 2 TO BB.
0030  PERFORM X10.
```

DELETE—to remove a line from the program in the work space

SYNTAX:	NNNN
WHERE:	NNNN is the line number of the line to be removed

Example:

```
?LIST 10-30
0010  MOVE 10 TO AA.
0020  ADD 1 TO BB.
0030  PERFORM X10.
?20
?LIST 10-30
0010  MOVE 10 TO AA.
0030  PERFORM X10.
```

The following advanced editing commands are also available to the user: CHANGE, APPEND, FIND, PRINT, DELETE, MOVE, DUPLICATE, BLOCK, VERIFY, NUMBER, MERGE and RESEQUENCE. A description of each follows:

CHANGE—to change a string within a line (or group of lines)

SYNTAX:	CHANGE /XXX/YYY/
	CHANGE NN/XXX/YYY/
	CHANGE NN-MM/XXX/YYY/
	CHANGE NN-MM;KK/XXX/YYY/
ALIAS:	C
WHERE:	"/" is a delimiter (may be any ASCII character except semicolon ";")
	NN is the first line to be checked
	MM is the last line to be checked
	KK is the occurrence number of strings within the line
	XXX is the string to be changed
	YYY is the string to be substituted
NOTE:	If NN-MM or NN is omitted, the entire workspace is searched and modified.
NOTE:	IF 'A' is used in the KK position, all occurrences will be changed.
NOTE:	The underline '_' is used as an ignore flag. If present in the XXX string, those character positions will be ignored when searching for a string, those characters will not be changed.
NOTE:	The ignore character may be changed by entering the following sequence C 0XY.
WHERE:	X is the old ignore character
	Y is the new ignore character

FIND—to search the current work space for a string

SYNTAX:	FIND /XXX/
	FIND NN-MM/XXX/

	FIND NN-MM;KK/XXX/
ALIAS:	F
WHERE:	"/" is a delimiter (may be any ASCII character except semicolon ";") NN is the first line to be checked MM is the last line to be checked KK is the occurrence counter of the string within the line XXX is the string to be found
NOTE:	FIND may be used to set the block range if block mode is in effect.
NOTE:	If 'A' is used in the KK field, all occurrences are displayed.
NOTE:	The underline '_' is interpreted as an ignore character in the XXX string.

APPEND—to add a string to the end of a line or group of lines

SYNTAX:	APPEND /XXX/ APPEND NN/XXX/ APPEND NN-MM/XXX/
ALIAS:	- A
DEFAULTS:	entire work space or block
WHERE:	NN is the first line number MM is the last line number XXX is the string to be added

Example:

```
?L 10
0010  MOVE 10 to AA.
?A 10/,BB/
0010  MOVE 10 to AA,BB
```

RESEQUENCE—to resequence the line numbers

SYNTAX:	RESEQUENCE RESEQUENCE MM RESEQUENCE MM, NN RESEQUENCE N (this will remove all line numbers)
ALIAS:	RSQ
WHERE:	MM is the new starting line number NN is the new increment
DEFAULT:	MM=10, NN=10
NOTE:	When line numbers are being removed the assumption is made that a blank follows the line number and this blank is also removed.
NOTE:	When in block mode, the block range is updated to reflect the new line numbers.

DUPLICATE—to duplicate a line (or group of lines) elsewhere in the work space

SYNTAX:	DUPLICATE NN,LL DUPLICATE NN-MM,LL DUPLICATE NN-MM,LL,KK
ALIAS:	D
WHERE:	NN is the first line to be duplicated MM is the last line to be duplicated LL is the new line number of the first record KK is the new line number increment (default = 1)
NOTE:	If LL exists the line or group of lines being duplicated will be placed before it.

MOVE—move lines or groups of lines elsewhere in the work space

SYNTAX:	MOVE NN,LL MOVE NN-MM,LL MOVE NN-MM,LL,KK
ALIAS:	M
WHERE:	NN is the key of the first line to be moved MM is the key of the last line to be moved LL is the new location of the first record KK is the line increment (default = 1)
NOTE:	If the LL line already exists, the moved line will be placed before it.
NOTE:	The move command may be used to resequence a group of lines by making NN and LL equal.
NOTE:	Unless LL is equal to NN, lines cannot be moved into the NN-MM range or record sequence errors will result.
NOTE:	Duplicate line number may be removed by using the move command to resequence the duplicate block.

BLOCK—change the edit mode from whole program to a group of lines or return to the whole program mode

SYNTAX:	BLOCK BLOCK OFF BLOCK IN BLOCK OUT
ALIAS:	B
WHERE:	IN, I or no operator envoke the BLOCK mode OFF, OUT, O are the out-of-effect (default) operators
NOTE:	When in the block mode the range of lines being operated on by the LIST, PRINT, APPEND FIND and CHANGE commands defaults to the last range specified by the LIST, PRINT, or FIND command.
NOTE:	The block mode is not applicable for files without line numbers.

VERIFY—the verify command may be used to turn on or off the verification printout after a successful change command execution

SYNTAX:	VERIFY VERIFY OFF VERIFY IN VERIFY OUT
ALIAS:	V
WHERE:	IN, I or no operator envoke the verify (default) OFF, OUT, O disable the verify

NUMBER—to be prompted with line numbers for new lines or to replace existing lines

SYNTAX:	NUMBER NUMBER MM, NN
ALIAS:	MM is the starting line number NN is the increment
DEFAULT:	MM=10, NN=10
NOTE:	A blank is automatically inserted after the line number. To exit the automatic number mode, enter a return key after the number is prompted.

DELETE—to delete lines

SYNTAX:	DELETE NN DELETE NN-MM
ALIAS:	DEL
WHERE:	NN is the first line to be removed MM is the last line to be removed

MERGE—to merge lines into the work space from another disk file

SYNTAX:	MERGE XXXX.SS:1 (NN-MM),LL MERGE XXXX.SS (NN-MM),LL MERGE XXXX(NN-MM),LL
ALIAS:	MRG
WHERE:	XXXX—disk file to be searched SS—disk file suffix (default=current file suffix) L—disk drive (default=current file drive) NN—line number of the first line to be copied MM—line number of the last line to be copied LL—line number in the work space after which the copied lines will be inserted
NOTE:	The LL line need not be present in the program and if not it will be created.
NOTE:	Lines are added to the program with line numbers starting with the LL line and incremented by one. If there exists a line number conflict the "RESEQUENCE NEEDED" message will be displayed at the console.

The following keyboard functions are also available as aids to the user.

RUBOUT/DEL—	displays the character being removed and deletes that character from the input stream.
CNTL+X—	gives a carriage return at the printer and deletes the current input line from the keyboard.
CNTL+W—	stops the current output; any other key will resume the output.
RETURN—	terminates the keyboard input and processes the current input buffer.
BREAK—	terminates the current output at the printer and returns control to the ready level.

CHAPTER 2
COMPILING THE COBOL SOURCE PROGRAM

The COBOL source program is compiled by entering the MDOS command "COBOL." The format of this command is:

```
COBOL FN;OPTIONS
```

Where: FN is the file name of a COBOL source program. The file must have a four-digit line number followed by a space starting each source line. The default file suffix is SA. The default disk number is zero.

Example:

```
COBOL PRTALL
COBOL PRTALL.SA
COBOL PRTALL.SA:0
```

Options are:

none	List only the source program errors on the console.
P	List only the source program errors on the printer.
S	List source to console as it is compiled.
L	List source and generated code to the console.
SP	List source to line printer as it is compiled.
LP	List source and generated code to the line printer.
O=	Output file name saves generated code which has been put in memory to the output file name for later execution. The default file suffix is LO and the default unit number is zero.
D	Debug mode—the compiled program will print each paragraph name on the printer during program execution.

In general, the options may be entered in any order; "SP" and "PS" are equivalent. However, the output file name (O=) must appear last.

Examples:

```
COBOL PAY
```

Compile the source program "PAY." If there are compiler errors display them on console.

```
COBOL PAY;SPO=PAY
```

Compile the source program "PAY" printing error messages, if any, and source on the line printer. Save the generated code in file "PAY.LO:O."

```
COBOL PAY;PDO=FKEY05
```

Compile the source program "PAY" printing error messages, if any, on the line printer. Save the generated code in file "FKEY05.LO:O." Trace coding will be generated which will print each paragraph name on the printer during execution.

If desired, the COBOL compilation process may be halted by depressing the "BREAK" key.

2.1.1 Source Program Listing Options

No list options no listing will be printed. If errors are detected, the line in error will be printed followed by the error message.

S Option Each COBOL source line will be listed. A slash in margin C of the source program may be used to start a line on a new page. The first L column printed is the hexadecimal address of the start of generated coding for the COBOL statement. This address may be used to locate execution-time errors. The hexadecimal sizes of the program (instructions) and data areas are printed at the end of the compilation.

L Option This option will print the generated code in addition to the information printed by the S option. This option is primarily intended for compiler problem reporting.

CHAPTER 3
EXECUTING THE COBOL PROGRAM

3.1 SYSTEM START-UP

3.1.1

The system can be configured such that at system start-up or power on it comes up in any of its two modes: MDOS, or RUN.

3.1.2

The system must be in run mode to execute a COBOL PROGRAM.

If the system is in MDOS MODE (prompt: equal sign), input "RUN" (carriage return) to enter the RUN mode.

If the system is in RUN mode, (prompt: "READY") it is ready to execute COBOL programs.

Example:

```
=RUN
(carriage return)
READY—(ready to execute COBOL programs)
```

3.2 MODES OF OPERATION

3.2.1 MDOS MODE

The MDOS prompt is an equal sign. It is used for program development and MOTOROLA-supplied utility functions. To run a utility function enter its name followed by a carriage return. To enter RUN mode prior to executing a COBOL problem enter "RUN" followed by a carriage return. A carriage return terminates all input in MDOS mode.

3.2.2 RUN MODE

The RUN prompt is "READY-."

This mode is used to run all COBOL application programs. The system can be returned to MDOS mode by entering "STOP" followed by a page send or by depressing shift function key 1 (hold shift key and press function key 1). The page send key terminates all input in RUN mode.

3.3 CALLING APPLICATIONS PROGRAMS

An applications program can be called by name (e.g., PAYROLL) or it can be assigned a function key. Files with names from "FKEY01" through "FKEY16" are automatically assigned to the respective function keys.

Files with names "SKEY03" to "SKEY16" are automatically assigned to the respective shift function keys. (Shift keys 1 and 2 are system reserved.) For example, to assign a program to function key 3, it would be given the name "FKEY03."

3.3.1 Calling by Name

Whenever the prompt "READY" is on the screen, an application can be called by entering its name followed by page send.

3.3.2 Calling by Function Key

Applications assigned to function keys can be called at any time the system is in RUN mode merely by pressing the function key. That is, the function key can be used with the prompt "READY" or it can terminate another application at any time it requests input, then run itself.

Shift function keys 1 and 2 are system reserved. Shift function key 1 returns the system to the MDOS level. Shift function key 2 returns the system to the "READY-" prompt any time the application program is requesting input.

3.4 TERMINATION OF AN APPLICATION

When an application program executes a STOP RUN, the screen is not cleared, but the prompt "READY-" is written on line #23. This allows the application to leave information on the screen for the operator.

3.5 SYSTEM MESSAGES

Application programs should not use lines #23 and #24 of the CRT. These are reserved for system prompts and messages.

System messages are of two types:

3.5.1 Operator Messages

These messages request some form of operator interaction such as inserting a new data disk or readying the printer. Some require an operator response to the terminal, Y for yes or N for no. The messages are listed in Appendix B.

3.5.2 Programmer Error Messages

The messages normally indicate a problem too serious for the operator such as a bug in the COBOL program or system. These messages have the following format:

```
** ERROR (XX) **
a brief descriptive message
AT (PC)
```

The message number is inserted at (XX) and the COBOL program counter when the error occurred is inserted at (PC). The program running when the error occurred is aborted. The messages are listed in Appendix B.

CHAPTER 4
FILE MANAGEMENT

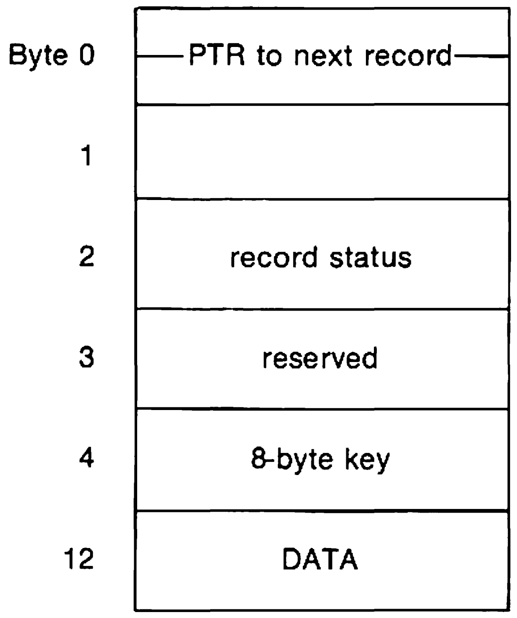
The file management system supports sequential and indexed sequential files. Facilities are available for creating these file types and for providing backup/recovery capability.

4.1 FILE DESCRIPTIONS

There is a maximum of eight files allowed per FMS data diskette. The files are designated by numbers 1 through 8. Space on the data diskette for a specific file is allocated one track (3328 bytes) at a time as it is needed. There is a total of 76 tracks available to be assigned to the files on the data diskette. The FMS data diskette usually resides on drive 1 of the floppy disk unit; however, in some cases it may reside on drive 0.

4.1.1 Indexed Sequential Files

The records of an indexed sequential file consist of 4 bytes of system information, an eight-byte key, and the data portion of the record. The record format is shown below.



The maximum size of the data and key portions of an indexed sequential record is 251 bytes.

4.1.2 Sequential Files

The records of a sequential file consist of 4 bytes of system information followed by the data portion of the record. The record format is shown below.

Byte 0	—PTR to next record—
1	
2	record status
3	reserved
4	DATA

4.2 DISKETTE INITIALIZATION

Before any files can be created on a diskette, the diskette must be cleared and initialized. This is accomplished on an MDOS system by putting a scratch diskette in drive 1 and typing the command "FMSDSK" at the MDOS command level. This program initialized a scratch diskette and prepares it for use as an FMS data diskette.

4.3 FILE CREATION

The files on an FMS data diskette are created using an interactive program which runs on an MDOS system. The program is brought into execution by typing the command "FMS" at the MDOS command level.

The following commands are supported by the FMS interactive program:

L or LIST—List file names and attributes.

C or CREATE—Create a specified file.

CREATE File Name, Record Size, File Type, Overflow records/track, duplicates option.

File Name—	The file name must be a number from 1 to 8.
Record Size—	The record size includes only the data and key portions of a record. The maximum record size is 251 bytes.
File Type—	I Indexed Sequential S Sequential

Overflow Records/Track—	This field applies only to indexed sequential files. It designates the number of record slots that are to be left at the end of each track for overflow when a backup of the data diskette is done. The maximum value for this field is 3328/(record size + 4).
Duplicates Option—	DUP This is an indexed sequential option which allows records with duplicate keys.

M or MODIFY—Modify the attributes of a specified file

MODIFY—	File Name, OVF Recs/Track (DUP) (NDP)
DUP—	Allow duplicate keys
NDP—	Do not allow duplicate keys

D or DELETE—Delete a specified file from the data diskette

DELETE—	File name

E or END—End execution of the interactive FMS program

4.4 BACKUP DATA (Drive One)

The backup function reorganizes the data on the original FMS data diskette (drive 1) and creates a new FMS data diskette (drive 0) with the reorganized data. It is more efficient to use the reorganized diskette during normal application processing since no delays will be encountered due to searching for records in overflow chains.

The backup function can be initiated by typing in the command "BACKUP1" while in the MDOS mode. Once initiated, the backup program instructs the user to remove the system diskette from drive 0 and insert a scratch diskette in drive 0. The backup program allows the user to supply a date and a sequence number. After this information is obtained, the scratch diskette in drive 0 is cleared and the files from drive 1 are copied to it. When all of the files have been copied, the user is instructed to remove the diskette in drive 0 and insert the system diskette. When the system diskette has been inserted in drive 0, control is returned to MDOS.

4.5 BACKUP SYSTEM (Drive Zero)

This backup function is used to back up the drive zero system disk. Since the information on the system disk does not change during normal day-to-day running of the application programs, it is only necessary to back up the system disk when the application programs are changed. The system backup function can be initiated by typing "BACKUP" while in the MDOS mode.

CHAPTER 5
GUIDE TO WRITING COBOL TRANSACTIONS

5.1 SCREEN DESIGN

5.1.1 Writing a Blank Form

This screen format is set up in the WORKING-STORAGE section. Data are sent to the screen by use of the DISPLAY command and data are read from the screen by use of the ACCEPT command.

Example:

```
WORKING-STORAGE Section
#1     01 SCREEN-1 LINE IS NEXT PAGE.
#2     02 FILLER PIC X(5) LINE 3 COLUMN 3 VALUE 'NAME'.
#3     02 FILLER PIC X(15) COLUMN 9 VALUE SPACE.
#4     02 FILLER PIC X (7) LINE 5, COLUMN 3; VALUE 'ADDRESS'.
#5     02 FILLER PIC X (20).
#6     01 RESPONSE.
#7     02 RNAME PIC X(15).
#8     02 RADDRESS PIC X(20).
          .
          .
          .
          .
PROCEDURE DIVISION.
#9     DISPLAY SCREEN-1.
#10    ACCEPT RESPONSE.
          .
          .
          .
          .
       END RUN.
```

In the preceding example on #1 "LINE IS NEXT PAGE" will cause the CRT screen to be cleared and all positions protected.

On #2 "LINE 3 COLUMN 3" will result in the word NAME displaying on line 3, starting at column 3.

On #3 "COLUMN 9" will move its position to column 9 staying on line 3 because a new line has not been stated. If the VALUE is a figurative constant (SPACE, ZERO) a field the size of the picture (15) is created. This field is unprotected and underscored. This will be a data-accepting field when displayed upon the CRT.

On #5 "LINE 5 COLUMN 3" will cause the word ADDRESS to start on line 5 at location 3.

#5 will create an unprotected field 20 characters in length starting 2 positions after the word ADDRESS. There will be one space between the word ADDRESS and the unprotected field.

The command DISPLAY SCREEN-1 will result in the following being displayed upon the CRT.

```
NAME    _______
ADDRESS ____________
```

5.1.2 Receiving Data From the Screen

Data can be entered only in the unprotected fields. The ACCEPT command would result in the data entered in the unprotected fields being returned to memory and residing in the fields with the labels RNAME AND RADDRESS.

5.1.3 Indicating a Field is in Error

The following technique can be used to blink and reverse video a protected field. This requirement is used to indicate a field discovered to be in error and that the user should correct and re-enter data on the screen.

The following tables are set up in WORKING-STORAGE.

```
77    X     PIC 9.
01    ERR—ROW.
      02 FILLER PIC 99 VALUE 3.
      02 FILLER PIC 99 VALUE 5.
01    ROW REDEFINES ERR-ROW PIC 99   OCCURS 2 TIMES.
01    ERR-COL.
      02 FILLER PIC 99 VALUE 15.
      02 FILLER PIC 99 VALUE 10.
01    COL REDEFINES ERR-COL PIC 99 OCCURS 2 TIMES.
```

If an error is discovered while editing the address field "RADDRESS," the following instructions would be performed in the PROCEDURE DIVISION.

```
        MOVE 2 TO X. (The address is the second data field.)
        PERFORM ERR-BLANK THRU ERR-BLINK-EDIT.
                            .
                            .
                            .
                            .
                            .
                            .
                            .
                            .
                            .

ERR-BLINK.
        DISPLAY     @(ROW(X), COL(X) ) $E0E2.

ERR-BLINK-EXIT.   EXIT.
```

The execution of this code would result in the address field blinking and reversing video.

The @(R,C) establishes the line and column location of the CRT.

A single $ precedes any of the CRT controlling codes.

```
        E0 sets the blink condition
        E2 sets the reverse video condition
```

The following instructions are used to turn off the blink and reverse video.

```
        MOVE 2 to X.
        PERFORM ERR-OFF THRU ERR-OFF-EXIT.

ERR-OFF.
        Display    @(ROW(X), COL(X) ), $E1E3.

ERR-OFF-EXIT.   EXIT.
        E1 sets the blink off
        E3 sets the reverse video off
```

The following is an example of a situation in which some data are always the same, such as titles, and other data are variable depending upon input from another source.

```
WORKING-STORAGE SECTION.
01   DATA-RECORD.
     02 PART-NUMBER    PIC X(4).
     02 DESC           PIC X(16).
     02 LOCATION       PIC 999.
     02 QTY-ON-HAND    PIC 9(4).
     02 COST           9(4)V99.
01   SCREEN-2 LINE IS NEXT PAGE.
     02 FILLER         PIC X(11)        LINE 4 COLUMN 2;
                                        VALUE 'DESCRIPTION'.
     02 SDESC          PIC X(16)        VALUE ' '.
     02 FILLER         PIC X(8)         LINE 6 COLUMN 2;
                                        VALUE 'LOCATION'.
     02 SLOC           PIC ZZ9          COLUMN 26; VALUE ' '.
     02 FILLER         PIC X(8)         LINE 8, COLUMN 2;
                                        VALUE 'QUANTITY'.
     02 SQTY           PIC XX9          VALUE ' '.
     02 FILLER         PIC X(4)         COLUMN 50;
                                        VALUE 'COST'.
     02 SCOST          PIC ZZZ.99       COLUMN 46 VALUE 'X'.
```

In the above layout the VALUE ' ' for SDESC, SLOC, SQTY and SCOST is used to prevent them from becoming unprotected fields.

```
PROCEDURE DIVISION
OPEN     DATAFILE.
READ     DATAFILE into DATA-RECORD.
```

```
MOVE     DESC TO SDESC.
MOVE     LOCATION TO SLOC.
MOVE     QTY-ON HAND TO SQTY.
MOVE     COST TO SCOST.
DISPLAY SCREEN-2.
```

The data is being read from a disk file and moved to the screen display. The screen is then displayed upon the CRT. There are no unprotected fields, i.e., there are no fields displayed on the CRT that can be changed and read back into memory.

5.2 ERROR OR OTHER MESSAGES

The following is an example of sending a message to the CRT. This message will go to line 24 column 3. Note: Never start on column 1 because the protected line code is in this position. Your data would overlay this code and the line would become unprotected. All of that line would be returned as data if the 'ACCEPT' verb was used.

```
DISPLAY @(24,3) $EOEAC4 'INVALID ENTRY' $EA.
```

The 'EO' sets the blink code.

The 'EO' thru 'EF' codes when used in a string, occupy a single position on the screen. Each code has a bit position within the byte.

Any time an 'E?' code is used in a location and the 'EA' (protect field) code is not already in this position the 'EA' should be used to prevent this field from becoming an unprotected field, unless that field is intended to be unprotected. This is the reason for the first 'EA,' following the 'EO' in the above example.

The 'C4' causes a move to the next position. If the C4 is not used, the I in INVALID ENTRY would overlay the EO and EA codes.

The final EA turns off the blink code so that nothing else on that line will blink and the rest of the line remains protected.

5.3 EDITING DATA FIELDS

The unprotected fields on the CRT will accept any type of input. Editing must be performed after the data are read back into memory to insure the correct type of data has been entered. All input fields from the CRT should be moved to another field set up in the proper format. The move allows many edit assisting features such as: ON SIZE ERROR, IF NUMERIC, IF ALPHABETIC, IF NEGATIVE, etc., tests to be utilized.

WORKING-STORAGE SECTION.

Example:

```
01  SCREEN-3 LINE IS NEXT PAGE.
    02 FILLER      PIC X(4) LINE 2, COLUMN 2 VALUE 'NAME'.
    02 S-NAME      PIC X(10) VALUE SPACE.
    02 FILLER      PIC X(8) LINE 4 COLUMN 2.
                            VALUE 'QUANTITY'.
```

```
              02 S-QTY        PIC X(4) COLUMN 12 VALUE SPACES.
              02 FILLER       PIC X(5) COLUMN 50 VALUE 'PRICE'.
              02 S-PRICE      PIC X(5) VALUE SPACE.
           01 SCREEN-IN.
              02 I-NAME       PIC X(10).
              02 I-QTY        PIC X(4).
              02 I-PRICE      PIC X(5).
           01 CLEAR-BLINK.
              02 FILLER       PIC X LINE 2, COLUMN 6 VALUE $E1.
              02 FILLER       PIC X LINE 4 COLUMN 11 VALUE $E1.
              77 E-NAME       PIC A(10).
              77 E-QTY        PIC S9(4).
              77 E-PRICE      PIC S999V99.
              77 ERROR-FLAG PIC 9 VALUE ZERO.


PROCEDURE DIVISION

      START.
            DISPLAY SCREEN-3.

      GET-SCREEN.
            ACCEPT      SCREEN-IN.
            PERFORM     BLINK-OFF.
            MOVE        I-NAME TO E-NAME
            IF          I-NAME EQUAL SPACES
                        PERFORM NAME-ERR GO TO EDIT-QTY.
            IF          E-NAME NOT ALPHABETIC
                        PERFORM NAME-ERR.

      EDIT-QTY.
            MOVE        I-QTY TO E-QTY
                        ON SIZE ERROR
                        PERFORM QTY-ERR GO TO EDIT-PRICE.
            IF          E-QTY IS NOT NUMERIC
                        PERFORM QTY-ERR GO TO EDIT-PRICE.
            IF          E-QTY IS NEGATIVE
                        PERFORM QTY-ERR GO TO EDIT-PRICE.
            IF          E-QTY IS GREATER THAN 10 PERFORM
                        QTY-ERR.

      EDIT-PRICE.
            MOVE        I-PRICE TO E-PRICE
                        ON SIZE ERROR
                        PERFORM PRICE-ERR GO TO CHECK-MORE.
            IF          E-PRICE IS NOT NUMERIC
                        PERFORM PRICE-ERR GO TO CHECK-MORE.
            IF          E-PRICE IS NEGATIVE
                        PERFORM PRICE-ERR.
```

```
CHECK-MORE.
          IF             ERROR-FLAG = 1
                         DISPLAY @(2,24) $E2EAC4 'INVALID ENTRY' $EA
                         GO TO GET SCREEN.
          NAME-ERR
                         DISPLAY @(2,6) $EO (Will cause field to blink.)
                         MOVE 1 TO ERROR-FLAG.
          NAME-ERR-EXIT.   EXIT.
          BLINK-OFF.
                         DISPLAY CLEAR-BLINK (Will result in blinks being reset).
                         MOVE ZERO TO ERROR-FLAG.
          BLINK-OFF-EXIT.   EXIT.
```

Edit rules for the above example.

The NAME field must not be blank and must have only alphabetic characters.

The QUANTITY field must be numeric only, positive, not greater than 10 and have no decimal positions.

The PRICE field must be numeric only, positive, no more than 3 positions to the left of the decimal point and no more than 2 decimal positions to the right of the decimal point.

The "ON SIZE ERROR" is used with a move in which the receiving field is defined as decimal or decimal with an implied decimal point.

Example:

```
01 X PIC 9(4).
(Will take the ON SIZE ERROR option if the sending field has a decimal
point.)
```

01 Y PIC 999V999 (Will take the ON SIZE ERROR option if the sending field has over 3 positions to the left of a decimal point or over 3 positions to the right of a decimal point.)

The following technique can be used to set up a date with slash separators.

```
01        DATE-REC PIC 9(6) VALUE 121078.
01        DATE-PRINT PIC ZZ/ZZ/ZZ.
          MOVE DATE-REC TO DATE-PRINT.
```

A display of DATE-PRINT after the MOVE would print 12/10/78. If DATE-REC contained zeros, only blanks would be contained in DATE-PRINT.

NOTE: When setting up CLEAR-BLINK (used to turn off the blink fields) the COLUMN location was one position before the field. The reason is that the codes (blink, unblink, etc.) always reside one position before the defined field.

5.4 WRITING TO THE PRINTER

```
FILE-CONTROL.
    SELECT PRTFILE
    ASSIGN TO PRINTER.
DATA DIVISION.
FILE SECTION.
FD PRTFILE.
    LINAGE IS 60
    TOP IS 6
    BOTTOM IS 0
    DATA RECORD IS PRINT-REC.
01 PRINT-REC PIC X(132).
PROCEDURE DIVISION.
    OPEN   OUTPUT   PRTFILE.
    MOVE   DATE   TO HOLD-DATE.
PRINT-LOOP.
    IF LINAGE-COUNTER EQUAL ZERO
      WRITE PRINT-REC FROM HEAD-LINE
    WRITE PRINT-REC FROM DETAIL-1
      AFTER ADVANCING 1 LINE.
    WRITE PRINT-REC FROM DETAIL-2
      AFTER ADVANCING 2 LINES. (Will double space)
      IF BREAK-KEY = 'Y' GO TO END-JOB.
```

The above example takes advantage of the automatic top of page being performed when LINAGE-COUNTER reaches 60. This is because LINAGE was defined as 60. The programmer performs the header routines when LINAGE-COUNTER is zero.

Another method of handling Top-of-Page is using the WRITE AFTER ADVANCING PAGE statement before the LINAGE-COUNTER reaches the maximum established in "LINAGE IS." The use of "AFTER ADVANCING" and "BEFORE ADVANCING" in the same program is not recommended.

The BREAK-KEY verb is used in a program to determine if the BREAK-KEY was activated. If so, the appropriate action should then be taken i.e., do a wrap-up and terminate the program. If BREAK-KEY = 'Y' (any valid statement). The statement would be executed if the BREAK-KEY had been depressed. The special register 'DATE' contains the date on the disk. If disk backup is provided daily and the current date is maintained on the disk, this feature provides a means at obtaining the current date. The current date is usually displayed on print reports.

TABLES

A Table is an arrangement of elements of one or more dimensions. COBOL tables may have from one to three dimensions. To refer to a specific element, the Table is subscripted or indexed. Tables are described by appending the OCCURS X TIMES clause to the data description. The X must be a positive integer greater than zero. Individual elements of a table cannot be assigned initial values. A record can be defined containing multipal data item with an initial value and then the record redefined as a Table.

```
01 V-RECORD.
    02 FILLER PIC   S99   VALUE 02.
    02 FILLER PIC   S99   VALUE 10.
    02 FILLER PIC   S99   VALUE 20.
01 TAB-I REDEFINES V-RECORD
    PIC   S99   OCCURS 3 TIMES.
77 SUB-1 PIC 9.
MOVE TAB-1(SUB-1) TO FLD-A.
```

The above example is a one-dimensional Table. TAB-1 must be subscripted or indexed to refer to a specific element. Subscripts must be positive, non-zero integer literals or identifiers that do not exceed the bounds of the Table.

A Table element can be indexed in addition to being subscripted. INDEXES are described by the INDEXED by phrase of the OCCURS clause.

```
01 TAB
    04      ITEM-T OCCURS 10 TIMES
            INDEXED BY I-1.
            06 ITEM-BASE PIC X(3) OCCURS 2 TIMES
            INDEXED by I-2.
    SET I-1 TO 10.
    SET I-2 TO 2.
    MOVE ITEM-BASE (I-1, I-2) TO WORK.
```

The above example is a two dimensional table using indexes. The last element of the table was moved to an area called WORK. INDEXES ARE manipulated by the SET statement.

SET I-1 to Value. The value may be an integer numeric literal, an identifier, another index or an index data item.
SET I-1 TO SUB-1.
SET I-1 UP BY 1.
SET I-1 DOWN BY 3.
SET HOLD TO I-1. (Identifier set to index value). Defining an index does not preclude the use of subscripts but they should not be used in combination. ITEM-BASE (I-1, SUB1) this is incorrect.

5.5 DISK I/O

There are two types of file structures, sequential and indexed.

5.5.1 ORGANIZATION SEQUENTIAL

ACCESS SEQUENTIAL
OPEN (INPUT) (OUTPUT) (EXTEND)

The OPEN must be executed before any other I/O command. The OPEN (INPUT) (OUTPUT) sets the pointer to the beginning of the file. The OPEN (EXTEND) sets the pointer to the end of the file.

CLOSE (FILE NAME) The CLOSE terminates processing of files and causes buffers to be written to the disk.

CLOSE (FILE NAME) WITH DELETE. This option of the CLOSE will delete all records from the file. This option is useful when the records processed are no longer needed. A backup DATA DISK should be in existence before running a program using this option in case a rerun is required.

READ (FILE NAME). The read will get the record the pointer is set up to retrieve. The Read used with the OPEN EXTEND is invalid.

WRITE (RECORD NAME). The record will be written to the location pointed to by the pointer. The WRITE used with the OPEN EXTEND option would write the record to the end of the file. This feature is useful if records are to be added to this file.

5.5.2 ORGANIZATION INDEXED

ACCESS SEQUENTIAL
OPEN (INPUT) (I/O) (OUTPUT)

The OPEN positions the pointer at the beginning of the file.

READ Records are read in ascending order of the Record Keys. (The next record in the file is read.)

START Will establish the pointer using the Record Key.

REWRITE Sequential updating by rewriting the last record read.

DELETE The last record read is marked for deletion. Actual record removal occurs at reorganization time.

ACCESS RANDOM
OPEN (INPUT) (I/O) (OUTPUT)

READ, WRITE, REWRITE and DELETE The Record Key must be provided. Access is to the record indicated by the key.

ACCESS DYNAMIC

READ, WRITE, REWRITE and DELETE Statements access the file just as they do for "ACCESS IS RANDOM."

The READ NEXT statement may be interspersed with these statements to read the file sequentially from the current file position. The file is positioned for sequential reading by use of the START statement.

CLOSE WITH DELETE option will delete all the records in the file.

RECORD KEY IS (Field Name) WITH DUPLICATES

This option is used for an INDEXED file when the need for duplicate keys is present.

An initial access will obtain the first record of a duplicate key. The other records are obtained by using a sequential statement such as READ NEXT. The programmer must determine if he actually obtained a duplicate record because the read gets the next record in the file with no regard to it being a duplicate.

The duplicate key feature is useful for a transaction file which is tied to a master file. All transactions for a master can be tied to that master by using the master key as transaction key. A random read to the trans file followed by READ NEXT would obtain all transactions for that master.

The START verb provides a means of positioning the pointer so that the next read will obtain the desired record. The start can be used with a partial key and blanks padded to the right.

```
MOVE 'AAA' TO PART.
START DATAFILE INVALID KEY GO TO NO FIND.
```

This example would position the pointer at the record whose key is 'AAAbbbb.' An exact match is required.

```
START DATAFILE KEY IS = PART
```

INVALID KEY GO TO NO FIND would be exactly the same as the above example.

```
START DATAFILE KEY IS GREATER PART INVALID KEY GO TO NO FIND.
```

The first record whose key is greater than 'AAA' would be pointed at. The invalid key return is taken if there is no key greater than 'AAA.'

```
START DATAFILE KEY IS NOT LESS PART
INVALID KEY GO TO NO FIND.
```

This statement would point at the part 'AAA' if it existed or any part number greater than 'AAA.'

A use for this command would be the processing of all records that fall in a specific range.

```
          MOVE 'C' TO PART.
          START DATAFILE KEY IS NOT LESS PART.
          INVALID KEY GO TO NO FIND.

LOOP
          READ DATAFILE NEXT AT END GO TO FILEEND.
          IF PART GREATER 'CZZZZZZZ' GO TO RANGEND.
          PERFORM PRINTIT.
          GO TO LOOP.
```

The above example could be used to process all the records whose key starts with the letter 'C.'

Example:

```
FILE-CONTROL.
        SELECT MASTER-1
            ASSIGN TO DISK INV:1
            ORGANIZATION IS RANDOM
            ACCESS IS RANDOM
            RECORD KEY IS PART-NUMBER.
        SELECT MASTER-2
            ASSIGN TO DISK INV:2
            ORGANIZATION IS RANDOM
            ACCESS IS SEQUENTIAL.
RECORD KEY IS PART-NAME.
        SELECT TRANFILE
            ASSIGN TO DISK INV:3
            ORGANIZATION IS SEQUENTIAL
            ACCESS IS SEQUENTIAL.
DATA DIVISION.
FILE SECTION
FD      MASTER-1
        LABEL RECORDS ARE OMITTED
        DATA RECORD IS MASTER-REC-1.
01      MASTER-REC-1.
02      PART NUMBER                 PIC X(18).
02      DESCRIPTION                 PIC X(10).
02      LOCATION                    PIC 9(4).
02      COST                        PIC 9(4)V99.
02      LAST-TRAN-DATE              PIC 9(6).
02      LAST-UPDATE-DATE            PIC 9(6).
02      TOTAL-TRAN                  PIC 9(5).
FD      MASTER-2
        LABEL RECORDS ARE OMITTED.
01      MASTER-REC-2.
        02  PART-NAME               PIC X(8).
        02  CROSS-CODE              PIC X(8).
        02  PART-NUMBER-2           PIC X(8).
        02  TOTAL-DOLLARS           PIC 9(5)V99.
        02  ORIGINAL-DATE           PIC 9(6).
FD      TRANFILE
        LABEL RECORDS ARE OMITTED.
01      TRAN-REC.
02      TRAN-NAME                   PIC X(10).
02      TRAN-QTY                    PIC 9(4).
02      TRAN-DATE                   PIC 9(6).
WORKING-STORAGE SECTION.
77      REL-COUNT                   PIC 9(4) VALUE ZERO.
77      HOLD-DATE                   PIC 9(6).
PROCEDURE DIVISION.
OPEN    I/O                         MASTER-1.
OPEN    INPUT                       MASTER-2 TRANFILE.
```

```
READ    MASTER-2   AT END GO TO DONE.
MOVE    DATE      TO HOLD-DATE.
READ-TRAN.
READ    TRANFILE AT END GO TO DONE.
COMP-REC.
IF      TRAN-NAME EQUAL PART-NAME GO TO USE-IT.
IF      TRAN-NAME LESS THAN PART-NAME GO TO READ-TRAN.
READ-MAS.
        READ MASTER-2 AT END GO TO DONE.
        GO TO COMP-REC.
USE-IT.
        MOVE PART-NUMBER-2 TO PART-NUMBER.
READ MASTER-1 INVALID KEY
        DISPLAY @(3,24) PART-NUMBER 'BAD NUMBER'.
        GO TO READ-MAS.
ADD     1 TO REC-COUNT.
ADD     TRAN-QTY TO TOTAL-TRAN.
MOVE    TRAN-DATE TO LAST-DATE.
MOVE    HOLD-DATE TO LAST-UPDATE-DATE.
REWRITE MASTER-1 INVALID KEY
        DISPLAY @(3,24) 'UPDATE DISK ERROR'
        CLOSE TRANFILE
        GO TO END.
DONE.
CLOSE TRANFILE WITH DELETE.
DISPLAY @(24,2) $D5 'RUN COMPLETED RECORDS UPDATED-'
        REC-COUNT.
END.
        CLOSE MASTER-1 MASTER-2.
        STOP RUN.
```

5.6 PROGRAMMING PROCEDURE

The PROCEDURAL STEPS TO BE CONSIDERED WHEN PROGRAMMING A DISPLAY/CHANGE

The first display message should request the key of the record to be displayed. The screen input is ACCEPTED and validated. The best validation is to read the disk using the supplied key. If the record is not present display an error message and allow them to enter another key.

The next step is to display the constant information. This display should also set up the unprotected fields where data are to reside. Move the variable data obtained from the disk record to the proper sequence for displaying upon the CRT.

Display the variable data. Because all positions on the screen are protected except the areas for your variable information the variable data go to the proper locations on the screen. The ACCEPT will read only the variable data back into memory.

Each field is now edited to insure that any changes conform to the edit rules.

Each field found to be in error is blinked or reverse video to help the user identify the incorrect fields.

An error message is sent to line 24 advising the user that some fields are in error. The program then transfers back to ACCEPT the variable data from the screen. All variable fields are re-edited whether they were in error or not the previous time. This is necessary because the user can change any unprotected field regardless of the previous data in the field.

The blink condition is turned off at the start of the edit processing. This will prevent old errors that have been corrected from continuing to blink. The simplest technique for turning off the blink is to turn off the blink for all unprotected fields regardless of their current condition.

After all fields are correct the data are moved back to the disk area and the disk record is written back to the disk.

A message is sent to the screen to inform the user that the transaction was successful.

The first display in a program should use LINE IS NEXT PAGE as part of its layout in WORKING-STORAGE. This will cause the screen to be cleared of any previous data and all positions protected. Unprotected fields that the programmer is unaware of (and therefore has not set up his program to handle) will cause invalid data to be returned with the next ACCEPT.

ERROR or successful completion messages are normally sent to line 24 column 2 or greater.

All messages or blink codes should be cleared after they have served their purpose to prevent confusion during later processing.

APPENDIX A. Compiler Error Messages

0	Illegal character has been used.
1	Continuation expected here
3	Parse stack overflow (compiler error)
4	Variable contains more than 30 characters or a number contains more than 15 digits.
5	Syntax error
7	The compiler has generated an out-of-range branch address.
8	COPY statements may not be nested.
9	File name used in a COPY statement cannot be found. The file name must end in SA.
10	This statement label has already been used.
11	This name is already defined.
12	Number of OCCURS is too large.
13	OCCURS is not allowed with FILLER.
14	This program is too big to execute.
15	The generated code and symbol table are overlapping in memory.
16	An invalid level number is being used.
17	REDEFINES statement is illegal.
18	PICTURE clause is too large.
19	The item previous to this statement is not a group. Therefore, this statement is out of order.
1A	Level 01 is missing.
1B	VALUE is not allowed on REDEFINES.
20	Too many statement labels are being used in a GOTO.
21	External name already used
22	Dummy argument name already used
23	Too many clauses in a PERFORM statement
24	Already resolved
30	Too many operands in an expression
31	The number of subscripts are not equal to the number of dimensions.
32	Subscript out of range
33	An undefined name has been used.
34	Operand must be an integer.
35	Name is not fully qualified.
36	More than three (3) subscripts are invalid.
37	A literal used with "ALL" must be only byte.
38	A literal used with "ALL" must not be numeric.
39	No OCCURS clause for the table
40	An unknown edit code has been used.
41	There is a conflict between numeric and alphanumeric.
42	A replacement code must be the leading character.
43	Only one edit sign may be used.
44	Only one V or period is allowed in a picture.
45	Only one S is allowed in an edit picture.
46	The characters CR and DB must be the last two characters of the edit field.
47	There is a conflict between USAGE and PICTURE.

48	There is a conflict between VALUE and PICTURE.
49	Justified RIGHT/PICTURE conflict
4A	BLANK when ZERO/PICTURE conflict
50	Too many nested IF statements
51	An incomplete logical expression has been found.
60	The RECORD KEY clause is missing.
61	The referenced file is not RANDOM.
62	An invalid device type is being used, or a disk file number is missing or invalid.
63	A line or column number is too large. Line max is 24, column max is 80.
64	A line or column overlaps last field.
65	This field overflows the screen.
66	A field cannot start in column one.
67	CRT cursor controls must be the first of any elementary item.
68	A level 77 cannot be used for cursor controls.
70	Operand must be numeric.
71	Arithmetic operation not recognized
72	Invalid logical expression
73	Operand cannot be a literal.
74	Compute statement requires too many intermediate results to be saved.

APPENDIX B. Execution Error Messages

Execution time operator and error messages

FILE ******** DOES NOT EXIST
: An undefined COBOL program or function key was requested.

LOAD SYSTEM DISK & REPLY
: Load a system disk into a drive zero and reply (send page is sufficient).

INSERT DISK ******** INTO DRIVE * (Y/N)
: Insert the named data disk into the requested drive, and reply with "Y" page send when it is done. A reply of "N" assumes the data disk is not available and the program is aborted.

PRINTER NOT READY—RETRY (Y/N) ?
: The printer is not ready. Ready it and reply 'Y' to retry, or reply N to abort the request and the program.

PROGRAM ERROR MESSAGES

The following are preceded by "ERROR (XX)" and followed by "AT (PC)" with the error number inserted at (XX) and the COBOL program counter in hexadecimal inserted at (PC).

00	UNDEFINED ERROR The error write routine received an undefined error number (probably a RUNTIME error and not COBOL).
03	INVALID FILE NAME An OPEN was attempted on an undefined file.
04	KEY OR LINK CHANGED The key or link in a record was changed between READ and REWRITE.
06	DISKETTE FULL Self-explanatory. Running a backup might make some spare room.
07	FILE NOT OPEN An FMS request was made to a file that is not OPEN.
08	CURRENT RECORD UNDEFINED No record is currently defined for an operation that expects one such as DELETE CURRENT RECORD.
09	INTERNAL FILE ERROR Something such as an illegal pointer causes FMS to think it is lost.
10	DISK ERROR Hardware disk error occurred.
11	ILLEGAL REQUEST THIS FILE TYPE The request is not legal for this file type.
15	ONLY ONE FILE CAN BE OPEN ON DRIVE 0 An attempt was made to OPEN a second file on drive 0.
17	DUPLICATES NOT ALLOWED ON THIS FILE This file does not allow duplicate keys.

18	RECORD LENGTHS DON'T AGREE The record length in the user's UCA does not agree with the length in the file directory.
19	DRIVE ONE HAS NON-DATA DISK The diskette in drive #1 is not a data disk.
30	ILLEGAL I-O CALL An illegal I-O call was made such as attempting to OPEN the printer for INPUT.
31	VERSIONS DON'T MATCH This COBOL program was compiled on a version of the compiler that is not compatible with this version of the runtime package.

APPENDIX C. Display Control Codes

HEXADECIMAL CODE	*COMMAND*
C0	CURSOR TO HOME POSITION
C1	CURSOR UP ONE LINE
C2	CURSOR DOWN ONE LINE
C3	CURSOR LEFT ONE COLUMN
C4	CURSOR RIGHT ONE COLUMN
C5	LOAD CURSOR POSITION
C6	READ CURSOR POSITION
C7	SET PAGE MODE
C8	SET SCROLL MODE
C9	SET TOP DISPLAY LINE
CA	SET LAST DISPLAY LINE
CB	SET LEFT DISPLAY COLUMN
CC	SET RIGHT DISPLAY COLUMN
CD	SET PROTECT MODE
CE	WRITE ABSOLUTE
CF	READ ABSOLUTE
D0	CHARACTER INSERT
D1	CHARACTER DELETE
D2	ENABLE KEYBOARD
D3	DISABLE KEYBOARD
D4	PAGE ERASE
D5	LINE ERASE
D6	LINE INSERT
D7	LINE DELETE
D8	CLEAR/HOME
D9	SEND PAGE
DA	TAB
DB	BACK TAB
DC	SET TABS
DD	START DATA
DE	END DATA
DF	SEND LINE
E0	SET BLINK
E1	RESET BLINK
E2	SET (FIELD) VIDEO INVERT
E3	RESET (FIELD) VIDEO INVERT
E4	SET HALF BRIGHT
E5	RESET HALF BRIGHT

HEXADECIMAL CODE	*COMMAND*
E6	SET UNDERLINE
E7	RESET UNDERLINE
E8	SET NON-DISPLAY
E9	RESET NON-DISPLAY
EA	SET FILE PROTECT
EB	RESET FIELD PROTECT
EC	SET TRANSPARENT MODE
ED	RESET TRANSPARENT MODE
EE	SET VIDEO INVERT-FULL SCREEN
EF	RESET VIDEO INVERT-FULL SCREEN
F1	TERMINAL RESET
F2	ALLOW STATUS INDICATORS
F3	DISALLOW STATUS INDICATORS
FA	ENABLE LOAD FUNCTION
FB	DISABLE LOAD FUNCTION
FC	SET DISPLAY SPECIAL CHARACTERS
FD	RESET DISPLAY SPECIAL CHARACTERS

HEXADECIMAL VALUE	*MEANING*
07	BELL
08	BACKSPACE
09	HORIZONTAL TAB
0A	LINE FEED
0B	VERTICAL TAB
0C	FORM FEED
0D	CARRIAGE RETURN
18	CANCEL
98	STATUS—PARITY ERROR
9A	STATUS—RECEIVED OVERRUN
9B	STATUS—FRAMING ERROR

APPENDIX D. CRT Switch Settings

There are three groups of option switches on the back of the CRT. These switches must be properly set for the COBOL page mode screen formatting to work properly.

GROUP ONE		GROUP TWO	
ENABLE	ON	DUPLEX	FULL
DISPLAY	OFF	PARITY	NO
TRANS MODE	OFF		EVEN
VIDEO INV	OFF	XMIT WORDS	8-BIT
A	ON	STOP BIT	1
B	OFF	CONNECTION	DIRECT
C	OFF	MODEM TYPE	103
SPEC CHAR	OFF	TURNAROUND	S-CHAN
LINE FREQ	OFF	CODE SEL	EOT

GROUP THREE

BAUD RATE—SELECT ONLY ONE. 9600 IS THE MOST DESIRABLE.

APPENDIX E. Sample COBOL Program

```
PAGE 001  SORT    .SA:0  APPENDIX E SAMPLE COBOL PROGRAM

0010 IDENTIFICATION DIVISION.
0020 PROGRAM-ID.  SORT
0030 *  THIS IS A MEMORY SORT ON KEYS. THE PROG HANDLES 2 KEYS.
0040 *  KEY-TABLE-1 IS MAJOR KEY FOR SORT.
0050 ENVIRONMENT DIVISION.
0060 SOURCE-COMPUTER. M6800.
0070 OBJECT-COMPUTER. M6800.
0080 INPUT-OUTPUT SECTION.
0090 FILE-CONTROL.
0100     SELECT DATAIN
0110         ASSIGN TO DISK DATADISK:1
0120         ORGANIZATION IS INDEXED
0130         ACCESS IS DYNAMIC
0140         RECORD KEY IS PART-NUMBER.
0150     SELECT DATAOUT
0160         ASSIGN TO DISK DATADISK:3
0170         ORGANIZATION IS SEQUENTIAL
0180         ACCESS IS SEQUENTIAL.
0190     SELECT PRINT-FILE
0200         ASSIGN TO PRINTER.
0210 *
0220 DATA DIVISION.
0230 FILE SECTION.
0240 FD DATAIN
0250     LABEL RECORDS ARE OMITTED
0260     DATA RECORD IS MASTER-REC.
0270  COPY MASREC.
0280 FD DATAOUT
0290    LABEL RECORDS ARE OMITTED
0300    DATA RECORD IS OUT-REC.
0310 01 OUT-REC PIC X(116).
0320 FD PRINT-FILE
0330     LINAGE IS 60
0340     TOP    IS 6
0350     BOTTOM IS 0.
0360 01 PRINT-REC PIC X(80).
0370 *
0380 WORKING-STORAGE SECTION.
0390 *  SORT-TAB-MAX = MAXIMUM SIZE OF KEY TABLES
0400 77 KEY-TABLE-1  PIC X(16) OCCURS 600 TIMES.
0410 77 KEY-TABLE-2  PIC X(8)  OCCURS 600 TIMES.
0420 77 SORT-TAB-MAX PIC 9(4)  VALUE 600.
0430 77 MAX          PIC 9(4).
0440 77 BASE-CENTER  PIC 9(4).
0450 77 WORK-CENTER  PIC 9(4).
0460 77 WORK-SEQ     PIC 9(4).
0470 77 POINT-1      PIC 9(4).
0480 77 TEMP1        PIC S9(4).
0490 77 KEY-TEMP     PIC X(15).
0500 77 POINT-2      PIC 9(4).
0510 77 ERR-FLAG     PIC 9     VALUE ZERO.
0520 77 PAGE-CNT     PIC 99    VALUE ZERO.
0530 01 HEAD.
0540   02 FILLER     PIC X(30) VALUE '  DESCRIPTION      PART-NUM'.
0550   02 FILLER     PIC X(19) VALUE ' QTY     LOC/BIN'.
0560   02 FILLER     PIC X(18) VALUE '              PAGE'.
0570   02 P-PAGE     PIC Z9.
0580 01 PRINT-DETAIL.
```

```
     02 P-DESC      PIC X(16).
     02 FILLER      PIC X(4)   VALUE SPACE.
     02 P-PART-NUM  PIC X(8).
     02 FILLER      PIC X(4)   VALUE SPACE.
     02 P-QTY       PIC ZZZ9.
     02 FILLER      PIC X(4)   VALUE SPACE.
     02 P-LOC       PIC ZZZ99.
*
PROCEDURE DIVISION.
*
AA-DRIVER.
    DISPLAY 'START OF RUN '.
    OPEN INPUT DATAIN.
    OPEN OUTPUT DATAOUT.
     OPEN OUTPUT PRINT-FILE.
    MOVE 1 TO MAX.
    DISPLAY 'START BUILD'.
    PERFORM BA-BUILD-FILE THRU BA-BUILD-FILE-EXIT.
    IF ERR-FLAG = 1 GO TO AA-END.
    MOVE MAX TO WORK-CENTER BASE-CENTER.
    DISPLAY 'START SORT'.
    PERFORM BB-SORT-IT THRU BB-SORT-IT-EXIT.
    MOVE 1 TO WORK-SEQ.
    DISPLAY 'START PRINT'.
    PERFORM BC-PROC-SORTED-KEY THRU BC-PROC-SORTED-KEY-EXIT.
    DISPLAY 'SORT ENDED'.
AA-END.
    CLOSE PRINT-FILE.
    CLOSE DATAOUT.
    CLOSE DATAIN.
    STOP RUN.
AA-DRIVER-EXIT.   EXIT.
*
*      *** THIS ROUTINE BUILDS INPUT FOR SORT ***
BA-BUILD-FILE.
    IF BREAK-KEY EQUAL 'Y'
       PERFORM CA-BREAK
       GO TO BA-BUILD-FILE-EXIT.
    READ DATAIN NEXT
       AT END
        COMPUTE MAX = MAX - 1
        GO TO BA-BUILD-FILE-EXIT.
    MOVE DESCRIPTION TO KEY-TABLE-1(MAX).
    MOVE PART-NUMBER TO KEY-TABLE-2(MAX).
    ADD 1 TO MAX.
    IF MAX GREATER SORT-TAB-MAX
       DISPLAY 'TABLE FULL RUN STOPPED'
       MOVE 1 TO ERR-FLAG
       GO TO BA-BUILD-FILE-EXIT.
    GO TO BA-BUILD-FILE.
BA-BUILD-FILE-EXIT.   EXIT.
*
*      *** THIS ROUTINE DOES THE SORTING ***
BB-SORT-IT.
    DIVIDE 2 INTO BASE-CENTER.
    IF BASE-CENTER EQUAL ZERO GO TO SORT-END.
DET-CENTER.
    SUBTRACT BASE-CENTER FROM MAX GIVING WORK-CENTER.
    MOVE 1 TO WORK-SEQ.
INIT-POINT-1.
    MOVE WORK-SEQ TO POINT-1.
TEST-FOR-TRANSFER.
    ADD POINT-1,BASE-CENTER GIVING POINT-2.
    IF KEY-TABLE-1(POINT-1) LESS THAN KEY-TABLE-1(POINT-2)
       GO TO CK-UPPER-LIMIT.
    IF KEY-TABLE-1(POINT-1) EQUAL TO KEY-TABLE-1(POINT-2)
       GO TO TEST-SORT-FLD2.
PERFORM-TRANS.
*      REVERSE POSITION OF KEYS IN TABLE
   MOVE KEY-TABLE-1(POINT-1) TO KEY-TEMP.
   MOVE KEY-TABLE-1(POINT-2) TO KEY-TABLE-1(POINT-1).
   MOVE KEY-TEMP TO KEY-TABLE-1(POINT-2).
*       MOVE 2ND TABLE
   MOVE KEY-TABLE-2(POINT-1) TO KEY-TEMP.
   MOVE KEY-TABLE-2(POINT-2) TO KEY-TABLE-2(POINT-1).
   MOVE KEY-TEMP TO KEY-TABLE-2(POINT-2).
NEXT-LOWER-SUBSC.
   SUBTRACT BASE-CENTER FROM POINT-1.
   IF POINT-1 > ZERO GO TO TEST-FOR-TRANSFER.
CK-UPPER-LIMIT.
   ADD 1 TO WORK-SEQ.
   SUBTRACT WORK-CENTER FROM WORK-SEQ GIVING TEMP1.
   IF TEMP1 > ZERO GO TO BB-SORT-IT.
   GO TO INIT-POINT-1.
TEST-SORT-FLD2.
*   REMOVE FOLLOWING INST IF SORT IS ONLY ON ONE KEY
   IF KEY-TABLE-2(POINT-1) > KEY-TABLE-2(POINT-2) GO TO PERFORM-TRANS.
   GO TO CK-UPPER-LIMIT.
SORT-END.
    DISPLAY 'END OF SORT PHASE'.
BB-SORT-IT-EXIT.    EXIT.
*
*     ***  ROUTINE CREATES OUTPUT AND PRINTING ***
BC-PROC-SORTED-KEY.
    MOVE KEY-TABLE-2(WORK-SEQ) TO PART-NUMBER P-PART-NUM.
    READ DATAIN  INVALID KEY
                     DISPLAY 'ERROR-UNABLE TO READ FILE'
                     GO TO BC-PROC-SORTED-KEY-EXIT.
    MOVE DESCRIPTION  TO P-DESC.
    MOVE QTY-ON-HAND  TO P-QTY.
    MOVE LOCATION-BIN TO P-LOC.
    IF LINAGE-COUNTER = ZERO
       ADD 1 TO PAGE-CNT
       MOVE PAGE-CNT TO P-PAGE
       WRITE PRINT-REC FROM HEAD
       WRITE PRINT-REC FROM PRINT-DETAIL AFTER ADVANCING 3 LINES
     ELSE
       WRITE PRINT-REC FROM PRINT-DETAIL AFTER ADVANCING 1 LINES.
*       CREATE SEQUENTIAL FILE IN SORT ORDER
*    WRITE OUT-REC FROM MASTER-REC.
    ADD 1 TO WORK-SEQ.
    IF WORK-SEQ > MAX
       NEXT SENTENCE
     ELSE
       IF BREAK-KEY = 'Y'
          PERFORM CA-BREAK
```

```
PAGE 004  SORT    .SA:0  APPENDIX E SAMPLE COBOL PROGRAM

1730          ELSE
1740             GO TO BC-PROC-SORTED-KEY.
1750 BC-PROC-SORTED-KEY-EXIT.   EXIT.
1760 CA-BREAK.
1770     DISPLAY 'BREAK KEY DEPRESSED-RUN STOPPED'
1780     MOVE 1 TO ERR-FLAG.
```

```
PAGE 001   ORDER   .SA:0  APPENDIX E SAMPLE COBOL PROGRAM

IDENTIFICATION DIVISION.
PROGRAM-ID. ORDER
AUTHOR.     MOTOROLA MICROSYSTEMS.
DATE-WRITTEN.    03/30/78.
DATE-COMPILED.   03/30/78
REMARKS.  THIS IS THE SET UP AN ORDER COMMAND.
*
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.  M6800.
OBJECT-COMPUTER.  M6800.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT DATAFILE
       ASSIGN TO DISK DATADISK:1
       ORGANIZATION IS INDEXED
       ACCESS IS RANDOM
       RECORD KEY IS PART-NUMBER.
    SELECT PRTFILE
       ASSIGN TO PRINTER.
*
DATA DIVISION.
FILE SECTION.
FD  DATAFILE
     LABEL RECORDS ARE OMITTED
     DATA RECORD IS MASTER-REC.
COPY MASREC.
FD   PRTFILE
     LABEL RECORD IS OMITTED
     DATA RECORD IS PRINT-REC.
01 PRINT-REC   PIC X(80).
*
WORKING-STORAGE SECTION.
77  ERRCK     PIC 9 VALUE 0.
77  E         PIC 99 VALUE 0.
77  X         PIC 99 VALUE 0.
77  Y         PIC 99 VALUE 0.
77 QTY-W      PIC S9(5).
77 COST-W     PIC S9(4)V99.
77 MOREA      PIC X.
01 H-DATE     PIC 9(6).
01 H-D-R REDEFINES H-DATE.
   02 H-MO    PIC 99.
   02 H-DAY   PIC 99.
   02 H-YR    PIC 99.
01 FUN.
   02 FILLER    PIC X(14) LINE IS NEXT PAGE COLUMN 34
                      VALUE 'PURCHASE ORDER'.
   02 FILLER PIC X(5)  COLUMN 60 VALUE 'DATE '.
   02 HOLD-DATE PIC 99/99/99 VALUE 0.
*
*   DISPLAY ITEM NUMBER REQUEST ON LINE 2 OF SCREEN
01  ITEM-NUM-LINE.
   02 FILLER    PIC X(4)    VALUE $C7C52121.
   02 FILLER    PIC X(11)   VALUE 'ITEM NUMBER'.
   02 FILLER    PIC X(6)    VALUE $C5212EEBE6C4.
   02 FILLER    PIC X(8)    VALUE ' '.
   02 FILLER    PIC XX      VALUE $EACD.
```

```
01 LINH.
  02 FILLER    PIC X(24) VALUE SPACE.
  02 FILLER     PIC X(19) VALUE 'PURCHASE ORDER FOR '.
  02 LINH-PART PIC X(8).
01 LIND.
  02 FILLER     PIC X(28) VALUE SPACE.
  02 LDESC      PIC X(16).
01 ANS.
  02 ANS-ITEM  PIC X(8).
*  ERRROW & ERRCOL ARE TABLE POS OF ROW & COL FOR
*  CURSOR FOR ERRORS
01  ERRROW.
  02 FILLER    PIC 99 VALUE 10.
  02 FILLER    PIC 99 VALUE 10.
  02 FILLER    PIC 99 VALUE 12.
01 ROW REDEFINES ERRROW PIC 99 OCCURS 3 TIMES.
01  ERRCOL.
  02 FILLER    PIC 99 VALUE 26.
  02 FILLER    PIC 99 VALUE 51.
  02 FILLER    PIC 99 VALUE 26.
01 COL REDEFINES ERRCOL PIC 99 OCCURS 3 TIMES.
01  ERR-TABLE.
  02 FILLER    PIC X(20) VALUE 'INVALID ENTRY
  02 FILLER    PIC X(20)  VALUE 'RECORD NOT FOUND
  02 FILLER    PIC X(20) VALUE '   SUCCESSFUL
01  ERR-MSG REDEFINES ERR-TABLE PIC X(20)
              OCCURS 3 TIMES INDEXED BY ERR-IND.
*    DATA FROM SCREEN
01 RSP.
  02 SNAM      PIC X(30).
  02 SVEN      PIC XXX.
  02 SADD      PIC X(30).
  02 SCITY     PIC X(20).
  02 SST       PIC X(5).
  02 SZIP      PIC X(5).
  02 SQTY      PIC X(5).
  02 SCOST     PIC X(7).
  02 SDATE.
    03 SMON    PIC 99.
    03 SDAY    PIC 99.
    03 SYR     PIC 99.
  02 SCOMM     PIC X(26).
01 LIN1.
  02 FILLER    PIC X(19) VALUE ' VENDOR NAME
  02 LNAM      PIC X(30).
  02 FILLER    PIC X(23) VALUE '     VENDOR NUMBER'.
  02 LVEN      PIC XXX.
01 LIN2.
  02 FILLER    PIC X(19) VALUE ' ADDRESS '.
  02 LADD      PIC X(30).
01 LIN3.
  02 FILLER    PIC X(19) VALUE ' CITY'.
  02 LCITY     PIC X(20).
  02 FILLER    PIC X(12) VALUE '    STATE'.
  02 LSTATE    PIC X(5).
  02 FILLER    PIC X(14) VALUE '      ZIP'.
  02 LZIP      PIC X(5).
01 LIN4.
```

```
   02 FILLER   PIC X(26) VALUE ' QUANTITY'.
   02 LQTY     PIC ZZZZ9.
   02 FILLER   PIC X(30) VALUE SPACE.
   02 FILLER   PIC X(7)  VALUE 'COST'.
   02 LCOST    PIC ZZZZ.99.
01 LIN5.
   02 FILLER   PIC X(23)  VALUE ' DATE OF ORDER'.
   02 LDATE    PIC ZZ/ZZ/ZZ.
   02 FILLER   PIC X(18)  VALUE '        COMMENT'.
   02 LCOMM    PIC X(26).
01 LIN6.
   02 FILLER   PIC X(55)  VALUE SPACE.
   02 FILLER   PIC X(12)  VALUE 'TOTAL COST'.
   02 LTOT     PIC ZZZZZ.99.
01 SCREEN.
   02 FILLER   PIC X      LINE 2  COLUMN 15  VALUE $EA.
   02 FILLER   PIC X(11) LINE 4  COLUMN 2   VALUE 'VENDOR NAME'.
   02 FILLER   PIC X(30)         COLUMN 20  VALUE SPACE.
   02 FILLER   PIC X(13)         COLUMN 57  VALUE 'VENDOR NUMBER'.
   02 FILLER   PIC XXX           COLUMN 73  VALUE SPACE.
   02 FILLER   PIC X(7)  LINE 6  COLUMN 2   VALUE 'ADDRESS'.
   02 FILLER   PIC X(30)         COLUMN 20  VALUE SPACE.
   02 FILLER   PIC X(4)  LINE 8  COLUMN 2   VALUE 'CITY'.
   02 FILLER   PIC X(20)         COLUMN 20  VALUE SPACE.
   02 FILLER   PIC X(5)          COLUMN 44  VALUE 'STATE'.
   02 FILLER   PIC X(5)          COLUMN 52  VALUE SPACE.
   02 FILLER   PIC XXX           COLUMN 63  VALUE 'ZIP'.
   02 FILLER   PIC X(5)          COLUMN 71  VALUE SPACE.
   02 FILLER   PIC X(8)  LINE 10 COLUMN 2   VALUE 'QUANTITY'.
   02 FILLER   PIC X(5)          COLUMN 27  VALUE SPACE.
   02 FILLER   PIC X(4)          COLUMN 44  VALUE 'COST'.
   02 FILLER   PIC X(7)          COLUMN 52  VALUE SPACE.
   02 FILLER   PIC X(8)  LINE 12 COLUMN 2   VALUE 'DATE OF '.
   02 FILLER   PIC X(13)                    VALUE 'ORDER(MMDDYY)'.
   02 FILLER   PIC X(6)          COLUMN 27  VALUE SPACE.
   02 FILLER   PIC X(7)  LINE 15 COLUMN 42  VALUE 'COMMENT'.
   02 FILLER   PIC X(26)         COLUMN 52  VALUE SPACE.
01    BLINK-OFF.
   02 FILLER   PIC X     LINE 10 COLUMN 26  VALUE $E3.
   02 FILLER   PIC X             COLUMN 51  VALUE $E3.
   02 FILLER   PIC X     LINE 12 COLUMN 26  VALUE $E3.
01 LOCK-MORE.
   02 FILLER   PIC X(17) VALUE $C7C52332EAC52367EAC52532EAC52732EA.
   02 FILLER   PIC X(16) VALUE $C52752EAC52765EAC52939EAC52952EA.
   02 FILLER   PIC X(12) VALUE $C52839EAC52E52EAC53722D5.
   02 FILLER   PIC X(4)  VALUE 'MORE'.
   02 FILLER   PIC X(6)  VALUE $EBE6C4C4EACD.
01 UNLOCK-S.
   02 FILLER   PIC X(5) VALUE $C7C5212EEA.
   02 FILLER   PIC X(16) VALUE $C52332EBC52367EBC52532EBC52732EB.
   02 FILLER   PIC X(16) VALUE $C52752EBC52765EBC52939EBC52952EB.
   02 FILLER   PIC X(9)  VALUE $C52839EBC52E52EBCD.
*****************************************************
PROCEDURE DIVISION.
AA-DRIVER-SECTION.
*
*  THIS SECTION WILL DO INITALIZATION OPEN CLOSE
*  PERFORM PROCESSING ROUTINES & WRAP UP
```

```
1750 *
1760     OPEN I-O DATAFILE.
1770     OPEN OUTPUT PRTFILE.
1780     MOVE SPACE TO MOREA.
1790 *     CLEAR & LOCK SCREEN
1800     MOVE DATE TO HOLD-DATE.
1810     DISPLAY FUN.
1820 *     GET PART NUM
1822 DISP-ITEM.
1825     DISPLAY ITEM-NUM-LINE.
1830 GET-ITEM.
1840     MOVE ZERO TO ERRCK.
1860     ACCEPT ANS.
1870 *     ERASE LINE 24
1880     DISPLAY @(24,2) $D5.
1890 *
1900     PERFORM BA-READ-RECORD THRU BA-READ-RECORD-EXIT.
1910     IF ERRCK EQUAL 1 GO TO GET-ITEM.
1920 *     DISPLAY BUILD REC SCREEN
1930 *
1940       IF MOREA EQUAL 'Y'
1950            DISPLAY UNLOCK-S
1960       ELSE
1970            PERFORM BB-FILL-SCREEN THRU BB-FILL-SCREEN-EXIT.
1980     PERFORM BC-DATA-EDIT THRU BC-DATA-EDIT-EXIT.
1990     PERFORM BD-DATA-UPDATE THRU BD-DATA-UPDATE-EXIT.
2000     DISPLAY LOCK-MORE.
2010     ACCEPT MOREA.
2020     IF MOREA EQUAL 'Y'
2030        DISPLAY @(24,2)$D5
2040        GO TO DISP-ITEM.
2050 *
2060 END-IT.
2070     CLOSE DATAFILE PRTFILE.
2080     STOP RUN.
2090 AA-DRIVER-EXIT.   EXIT.
2100 *
2110 *
2120 BA-READ-RECORD.
2130 *
2140 *   THIS ROUTINE WILL VALIDATE THAT THE PART IS IN THE FILE
2150 *
2160     MOVE ANS-ITEM TO PART-NUMBER.
2170     READ DATAFILE
2180         INVALID KEY
2190          SET ERR-IND TO 2
2200       PERFORM MSG-PRT THRU MSG-PRT-EXIT.
2210 BA-READ-RECORD-EXIT.    EXIT.
2220 *
2230 *
2240 BB-FILL-SCREEN.
2250     DISPLAY SCREEN.
2260 BB-FILL-SCREEN-EXIT.   EXIT.
2270 *
2280 *
2290 BC-DATA-EDIT.
2300 *
2310 *   THIS ROUTINE WILL READ SCREEN & VALIDATE DATA FIELDS
```

```
2320     ACCEPT RSP.
2330     DISPLAY BLINK-OFF.
2340     DISPLAY @(24,3) $D5.
2350     MOVE ZERO TO ERRCK Y.
2360 *
2370     MOVE SQTY  TO QTY-W ON SIZE ERROR GO TO QTY-ERR.
2380     IF QTY-W NOT NUMERIC GO TO QTY-ERR.
2390     IF QTY-W POSITIVE GO TO CK-COST.
2400 QTY-ERR.
2410     MOVE 1 TO X.
2420     PERFORM ERR-BLINK THRU ERR-BLINK-EXIT.
2430 CK-COST.
2440     MOVE SCOST TO COST-W ON SIZE ERROR GO TO COST-ERR.
2450     IF COST-W NOT NUMERIC GO TO COST-ERR.
2460     IF COST-W POSITIVE GO TO CK-DATE.
2470 COST-ERR.
2480     MOVE 2 TO X.
2490     PERFORM ERR-BLINK THRU ERR-BLINK-EXIT.
2500 CK-DATE.
2510     MOVE SDATE TO H-DATE ON SIZE ERROR GO TO DATE-ERR.
2520     IF H-DATE NOT NUMERIC GO TO DATE-ERR.
2530     IF H-MO GREATER THAN 12 GO TO DATE-ERR.
2540     IF H-DAY GREATER THAN 31 GO TO DATE-ERR.
2550     IF H-YR GREATER THAN 77 GO TO CK-ANY-ERR.
2551 *    GO TO CK-ANY-ERR.
2560 DATE-ERR.
2570     MOVE 3 TO X.
2580     PERFORM ERR-BLINK THRU ERR-BLINK-EXIT.
2590 CK-ANY-ERR.
2600     IF ERRCK EQUAL 1
2610        SET ERR-IND TO 1
2620        ADD 5 TO Y
2630        PERFORM MSG-PRT THRU MSG-PRT-EXIT
2640        PERFORM TAB-IT Y TIMES
2650        GO TO BC-DATA-EDIT.
2660     MOVE SNAM  TO LNAM.
2670     MOVE SVEN  TO LVEN.
2680     MOVE SADD  TO LADD.
2690     MOVE SCITY TO LCITY.
2700     MOVE SST   TO LSTATE.
2710     MOVE SZIP  TO LZIP.
2720     MOVE QTY-W TO LQTY.
2730     MOVE COST-W TO LCOST.
2740     MOVE H-DATE TO LDATE.
2750     MOVE SCOMM  TO LCOMM.
2760     MULTIPLY QTY-W BY COST-W GIVING LTOT.
2770     MOVE PART-NUMBER TO LINH-PART.
2780     MOVE DESCRIPTION TO LDESC.
2790 *
2800 *   PRINT ORDER
2810      IF MOREA EQUAL 'Y'
2820        WRITE PRINT-REC FROM LINH AFTER ADVANCING PAGE
2830      ELSE
2840        WRITE PRINT-REC FROM LINH.
2850     MOVE LIND TO PRINT-REC.
2860     PERFORM PRT-IT THRU PRT-IT-EXIT.
2870     WRITE PRINT-REC FROM LIN1 AFTER ADVANCING 2 LINES.
2880     MOVE LIN2 TO PRINT-REC.
```

```
2890      PERFORM PRT-IT THRU PRT-IT-EXIT.
2900      MOVE LIN3 TO PRINT-REC.
2910      PERFORM PRT-IT THRU PRT-IT-EXIT.
2920      MOVE LIN4 TO PRINT-REC.
2930      PERFORM PRT-IT THRU PRT-IT-EXIT.
2940      MOVE LIN5 TO PRINT-REC.
2950      PERFORM PRT-IT THRU PRT-IT-EXIT.
2960      WRITE PRINT-REC FROM LIN6 AFTER ADVANCING 2 LINES.
2970      ADD QTY-W TO QTY-ON-ORDER.
2980      IF ORDER-MONTH EQUAL ZERO
2990      MOVE H-DATE TO ORDER-DATE.
3000 BC-DATA-EDIT-EXIT.    EXIT.
3010 *
3020 *
3030 ERR-BLINK.
3040 *   ROUTINE TO BLINK FIELD IN ERROR
3050 *
3060      DISPLAY $C7 @(ROW(X), COL(X))$E2CD
3070      IF Y EQUALS ZERO MOVE X TO Y.
3080      MOVE 1 TO ERRCK.
3090 ERR-BLINK-EXIT.    EXIT.
3100 *
3110 *
3120 BD-DATA-UPDATE.
3130 *
3140 *   THIS ROUTINE WILL WRITE THE MASTER TO DISK
3150 *
3160      REWRITE MASTER-REC INVALID KEY
3170         SET ERR-IND TO 2
3180         PERFORM MSG-PRT THRU MSG-PRT-EXIT
3190         GO TO BD-DATA-UPDATE-EXIT.
3200 *   SUCCESSFUL MSG
3210      SET ERR-IND TO 3.
3220      PERFORM MSG-PRT THRU MSG-PRT-EXIT.
3230 BD-DATA-UPDATE-EXIT.   EXIT.
3240 *
3250 *
3260 MSG-PRT.
3270 *
3280 *   ROUTINE TO PRINT MESSAGES ON SCREEN
3290 *
3300      DISPLAY @(24,3) $EAE0C4 ERR-MSG(ERR-IND).
3310      MOVE 1 TO ERRCK.
3320 MSG-PRT-EXIT.   EXIT.
3330 *
3340 *
3350 TAB-IT.
3360    DISPLAY $09.
3370 TAB-IT-EXIT.   EXIT.
3380 *
3390 PRT-IT.
3400      WRITE PRINT-REC AFTER ADVANCING 2 LINES.
3410 PRT-IT-EXIT.   EXIT.
```

```
PAGE 001  MASREC  .SA:0  APPENDIX E SAMPLE COBOL PROGRAM

0010 01  MASTER-REC.
0015    02 PART-NUMBER     PIC X(8).
0020    02 DESCRIPTION     PIC X(16).
0030    02 LOCATION-BIN    PIC 9(5).
0040    02 COST            PIC 9(4)V99.
0050    02 LIST-PRICE      PIC 9(4)V99.
0060    02 TRADE-PRICE     PIC 9(4)V99.
0070    02 QTY-ON-HAND     PIC 9(5).
0080    02 QTY-ON-ORDER    PIC 9(5).
0090    02 QTY-PER-PACK    PIC 999.
0100    02 REORDER-POINT   PIC 9(5).
0110    02 STOCKING-QTY    PIC 9(5).
0120    02 VENDOR-CODE     PIC 999.
0130    02 LEAD-TIME       PIC 999.
0140    02 ORDER-DATE.
0150       03 ORDER-MONTH  PIC 99.
0160       03 ORDER-DAY    PIC 99.
0170       03 ORDER-YEAR   PIC 99.
0180    02 ISSUE-COUNT.
0190       03 ISS-MONTH-1  PIC 9(5).
0200       03 ISS-MONTH-2  PIC 9(4).
0210       03 ISS-MONTH-3  PIC 9(4).
0220       03 ISS-QUARTER-1 PIC 9(4).
0230       03 ISS-QUARTER-2 PIC 9(4).
0240       03 ISS-QUARTER-3 PIC 9(4).
0250    02 BACK-ORDER-IND PIC X.
0260    02 COMMENT         PIC X(8).
```

-COBOL SOFTWARE PROBLEM REPORT-

DATE:

NAME:

ADDRESS:

PHONE NUMBER:

PROBLEM DESCRIPTION:

RETURN THIS FORM TO MOTOROLA MICROSYSTEMS, PHOENIX, ARIZONA.

INCLUDE A LISTING OF THE PROGRAM (S OR L OPTION)
ALONG WITH A DUMP OF THE GENERATED OBJECT (LO FILE) MODULE.

ALSO INCLUDE ANY OTHER INFORMATION THAT MAY BE APPROPRIATE
TO THE SOLUTION OF THE PROBLEM.

SUGGESTION/PROBLEM REPORT FORM

M6800
RESIDENT COBOL
OPERATIONS REFERENCE MANUAL
COBOL 1.0

Motorola welcomes your comments on its software products and publications. Please use the back of this postpaid form. Should you wish to report a software problem, however, please send (to the address on the business reply portion of this form) a listing of the program (S or L option), a dump of the generated object (LO file) module and a complete description of the problem and/or solution.

Fold on this line

FIRST CLASS
Permit No. 2565
PHOENIX
ARIZONA

BUSINESS REPLY MAIL
No Postage stamp necessary if mailed in The United States

POSTAGE WILL BE PAID BY

Motorola Microsystems
3102 North 56th Street
Phoenix, AZ 85018

Attention: Publications Manager

Fold on this line

Please
Print

Name	Title	
Company	Division	
Street	Mail Drop	Phone Number
City	State	Zip

COMMENTS

Fold on indicated lines, tape or staple and mail.

MOTOROLA INC.

Integrated Circuits Division

Microsystems • 3102 North 56th Street • Phoenix, Arizona 85018

[illegible]