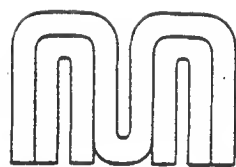


**DRAFT**

**CIS COBOL**

**OPERATING GUIDE**

**VERSION 4**



**MICRO FOCUS**

CIS COBOL  
OPERATING GUIDE  
For Use With the  
BASF 7100 Series BOS Operating Systems  
Version 4

**DRAFT**

Micro Focus Limited

Release 3  
October 1979

Not to be copied without the consent of Micro Focus Ltd.

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 herewith.

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

FLOW-MATIC (Trademark for Sperry Rand Corporation) Programming for the Univac (R) I and II, Data Automation Systems copyrighted 1958, 1959, by Sperry Rand Corporation; IBM Commercial Translator Form No. F28-8013, copyrighted 1959 by IBM; FACT, DS127A5260-2760, copyrighted 1960 by Minneapolis-Honeywell.

have specifically authorised 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.

**DRAFT**

BOS is a trademark of BASF

 **MICRO FOCUS**

Micro Focus Ltd.  
58, Acacia Road,  
St. Johns Wood,  
London NW8 6AG  
Telephone: 01 722 8843  
Telex: 28536 MICROF G

© COPYRIGHT 1980 by Micro Focus Ltd.

COBOL Release 4.4  
~~~~~

A new Release of CIS-COBOL is now available. All known errors occurring in 4.3 have been removed, some new features (e.g. symbolic RTS error messages) have been added.

A special subroutine has been integrated by BASF. This routine enables a COBOL program to:

- execute all FCS commands directly from COBOL (copyfile, copydisk, erase,...)
- call MTX/80

These two functions can be used too to load self-written Assembler programs and execute them either as subroutine (via ABSCAL) or even as additional tasks. An available storage area for such routines is from 23F0 to 29D4.


A demonstration program can be found on the distributed diskette (IFACE.CBL). This program can be deleted at any time. The same is true for program ASYNC which is the called Assembler subroutine.

On the 4.3 COBOL disk you found four FORMS2 modules too (CH1, CH2, GN1, GN2). Thus a CHECKOUT or GEN program - created by FORMS2 - could be compiled without having the FORMS2 disk inserted. Since on the 4.4 disk we have demonstration programs there was no space left for the FORMS2 routines. So if you want the exactly same COBOL disk: erase (or copy to some other disk) these demo programs, then copy the above FORMS2 routines onto the 4.4 COBOL/RTS disk.

CIS COBOL OPERATING GUIDE

VERSION 4

AMENDMENT RECORD

| Amendment Number | Dated    | Inserted by | Signature                                                                          | Date     |
|------------------|----------|-------------|------------------------------------------------------------------------------------|----------|
| Vmsii 4.4        | 14-09-81 | J.A. Kentii |  | 14-09-81 |

## PREFACE

This manual describes operating procedures for the BOS resident releases of the CIS COBOL Compiler and run-time libraries. The compiler converts CIS COBOL source code into an intermediate code which is then interpreted by the Run-Time System. The manual describes the steps needed to compile a program and then execute the compiled program, including all necessary linkage, relocation, and run-time requirements. Operation of the run-time Debug package is also included. A Screen Formatter program known as FORMS is also available with CIS COBOL and its operation is described in a separate manual.

## MANUAL ORGANIZATION

Chapters 1 through 4 of this manual describe compiler features and general procedures for linking, loading and execution of programs. Chapter 5 describes the operation of the configuration utility program CONFIG and Chapter 6 describes the use of source output from the screen formatter program FORMS in a CIS COBOL source program.

The appendices provide summarized information for reference purposes and give configuration information for various run-time environments.

## AUDIENCE

This manual is intended for personnel already familiar with COBOL usage on other equipment.

This manual contains the following chapters and appendices:

"Chapter 1. Introduction", which gives a general description of the CIS COBOL system, its input and output files, and the run-time libraries provided with the compiler, plus the step-by-step outline of compilation, linking, locating and executing of sample interactive programs.

"Chapter 2. Compiler Controls", which describes compiler commands, directives and listing formats.

"Chapter 3. Run Time System Controls", which gives general instructions for running programs, console operation, CRT screen handling and interactive debugging.

"Chapter 4. Multilanguage CALL Facilities", which describes the facilities available to invoke other COBOL programs or programs written in other languages from a main program.

"Chapter 5. Incorporating FORMS Utility Output", which describes the use of the FORMS screen formatting utility programs output.

"Appendix A. Summary of Compiler and Run Time Directives", summarises the compiler directives available in the CIS COBOL compiler.

"Appendix B. Compile-Time Errors", which lists all errors that can be signalled during program compilation.

"Appendix C. Run-Time Errors", which lists all errors that can be signalled during program execution.

"Appendix D. Operating Systems Errors", which is a listing of the error messages issued by the BOS Operating System.

"Appendix E. Interactive Debug Command Summary", which summarises the commands that can be used with the CIS COBOL Interactive Debug program.

"Appendix F. BOS Disk Files", which is a description of file naming conventions and formats used by CIS COBOL under BOS.

"Appendix G. Example of User Run-Time Subroutines", which gives sample user-programmed run-time subroutines.

"Appendix H. Supplied CALL Code Routines Example", which is a typical print-out showing calls to supplied sample subroutines.

#### NOTATION IN THIS MANUAL

Throughout this manual the following notation is used to describe the format of data input or output:

1. All words printed in small letters are generic terms representing names which will be devised by the programmer.
2. When material is enclosed in square brackets [ ], it is an indication that the material is an option which may be included or omitted as required.
3. The symbol << after a CRT entry or command format in this manual indicates that the data input terminator keys End of Message and Enter must be pressed to enter the command, except for Interactive Debug commands where it means press the Enter (without End of Message) or Return.

Headings are presented in this manual in the following order of importance:

CHAPTER n }  
TITLE } Chapter Heading

ORDER ONE HEADING  
ORDER TWO HEADING  
Order Three Heading  
Order Four Heading } Text 3 lines down

Order Five Heading: Text on same line

Numbers one (1) to nine (9) are written in text as letters e.g. one.  
Numbers ten (10) upwards are written in text as numbers e.g. 12

Bars in the right hand margin indicate changes since Version 4 Release 2.

RELATED PUBLICATIONS

For details of the CIS COBOL Language, refer to the document:

CIS COBOL Language Reference Manual

For details of the BOS Operating System, Messages, and File Structures refer to the BOS User manual.



## TABLE OF CONTENTS

### CHAPTER 1

#### INTRODUCTION

|                                               |     |
|-----------------------------------------------|-----|
| <u>GENERAL DESCRIPTION</u>                    | 1-1 |
| <u>GETTING STARTED WITH CIS COBOL</u>         | 1-1 |
| ISSUE DISK                                    | 1-1 |
| THE COMPILER                                  | 1-2 |
| THE RUN TIME SYSTEM                           | 1-2 |
| THE FORMS PROGRAM                             | 1-2 |
| THE DEMONSTRATION PROGRAMS                    | 1-2 |
| THE RUN TIME SUBROUTINES                      | 1-2 |
| FIRST STEPS                                   | 1-2 |
| <u>Initialisation</u>                         | 1-3 |
| <u>Disk Initialisation</u>                    | 1-3 |
| <u>Compilation</u>                            | 1-3 |
| <u>Running the Demonstration Programs</u>     | 1-4 |
| Calculation of $\pi$ (PI)                     | 1-4 |
| Stock Control Program One<br>(Cursor Control) | 1-4 |
| Stock Control Program Two<br>(Data Input)     | 1-5 |
| <u>PROGRAM DEVELOPMENT CYCLE</u>              | 1-5 |

### CHAPTER 2

#### COMPILER CONTROLS

|                            |     |
|----------------------------|-----|
| <u>COMMAND LINE SYNTAX</u> | 2-1 |
| <u>COMPILER DIRECTIVES</u> | 2-1 |
| ANS                        | 2-1 |
| RESEQ                      | 2-1 |
| NOINT                      | 2-1 |
| NOLIST                     | 2-1 |
| COPYLIST                   | 2-2 |
| NOFORM                     | 2-2 |
| ERRLIST                    | 2-2 |
| LIST                       | 2-2 |
| FORM                       | 2-2 |
| NOECHO                     | 2-2 |
| EXCLUDED COMBINATIONS      | 2-2 |

|                                   |     |
|-----------------------------------|-----|
| <u>SUMMARY INFORMATION ON CRT</u> | 2-3 |
| <u>LISTING FORMATS</u>            | 2-4 |

### CHAPTER 3

#### RUN TIME SYSTEM CONTROLS

|                                            |      |
|--------------------------------------------|------|
| <u>RUN TIME DIRECTIVES</u>                 | 3-1  |
| COMMAND LINE SYNTAX                        | 3-1  |
| <u>Load Parameters</u>                     | 3-1  |
| <u>Switch Parameter</u>                    | 3-2  |
| <u>Standard ANSI COBOL Debug</u>           |      |
| <u>Switch Parameter</u>                    | 3-2  |
| <u>Link Parameter</u>                      | 3-2  |
| <u>Program Parameters</u>                  | 3-3  |
| COMMAND LINE EXAMPLES                      | 3-3  |
| <u>INTERACTION IN APPLICATION PROGRAMS</u> | 3-4  |
| CRT SCREEN HANDLING                        | 3-4  |
| <u>Screen Layout and Format Facilities</u> | 3-4  |
| <u>Cursor Control Facilities</u>           | 3-5  |
| <u>INTERACTIVE DEBUGGING</u>               | 3-6  |
| THE P COMMAND                              | 3-6  |
| THE G COMMAND                              | 3-7  |
| THE X COMMAND                              | 3-7  |
| THE A COMMAND                              | 3-8  |
| THE S COMMAND                              | 3-9  |
| THE '.' COMMAND                            | 3-9  |
| THE T COMMAND                              | 3-9  |
| DEBUG MACRO COMMANDS                       | 3-10 |
| <u>The L Command</u>                       | 3-10 |
| <u>The \$ Command</u>                      | 3-10 |
| <u>The C Command</u>                       | 3-10 |
| <u>The ; Command</u>                       | 3-10 |

### CHAPTER 4

#### MULTILANGUAGE CALL FACILITIES

|                              |     |
|------------------------------|-----|
| <u>INTRODUCTION</u>          | 4-1 |
| <u>COBOL PROGRAMS - CALL</u> | 4-1 |
| THE RUN UNIT                 | 4-1 |
| FORMAT OF CIS COBOL "CALL"   | 4-1 |
| FORM OF CIS COBOL PROGRAMS   | 4-1 |
| RUN TIME PROGRAM LINKAGE     | 4-1 |
| EXAMPLE LINKAGE              | 4-2 |
| LIMITATIONS OF CALL          | 4-2 |

|                                    |     |
|------------------------------------|-----|
| <u>RUN TIME SUBROUTINES - CALL</u> | 4-3 |
| USER SUBROUTINES                   | 4-3 |
| ASSEMBLER SUBROUTINES PROVIDED     | 4-3 |
| <u>The CHAIN Subroutine</u>        | 4-4 |
| <u>The PEEK Subroutine</u>         | 4-5 |
| <u>The GET Subroutine</u>          | 4-6 |
| <u>The PUT Subroutine</u>          | 4-7 |
| <u>The ABSCAL Subroutine</u>       | 4-8 |

## CHAPTER 5

### INCORPORATING FORMS UTILITY PROGRAM OUTPUT

|                                                 |     |
|-------------------------------------------------|-----|
| <u>INTRODUCTION</u>                             | 5-1 |
| <u>SCREEN LAYOUT FACILITY</u>                   | 5-1 |
| MAJOR FACILITIES                                | 5-1 |
| CIS COBOL PROGRAMMING FOR FORMS SCREEN LAYOUT   | 5-1 |
| <u>GENERATED PROGRAMS</u>                       | 5-2 |
| CIS COBOL PROGRAMMING FOR FORMS GENERATED FILES | 5-2 |

## APPENDIX A

### SUMMARY OF COMPILER DIRECTIVES

## APPENDIX B

### COMPILE TIME ERRORS

## APPENDIX C

### RUN TIME ERRORS

## APPENDIX D

### OPERATING SYSTEM ERRORS

## APPENDIX E

### INTERACTIVE DEBUG COMMAND SUMMARY

## APPENDIX F

### BOS DISK FILES

APPENDIX G

EXAMPLE OF USER RUN TIME SUBROUTINES

APPENDIX H

SUPPLIED CALL CODE ROUTINES  
EXAMPLE

TABLES

| <u>Table</u> | <u>Title</u>                        | <u>Page</u> |
|--------------|-------------------------------------|-------------|
| 1-1          | Issue Disk Contents                 | 1-1         |
| 2-1          | Excluded Combinations of Directives | 2-3         |
| 3-1          | CRT Cursor Control Keys             | 3-5         |

## CHAPTER 1

### INTRODUCTION

#### GENERAL DESCRIPTION

COBOL (Common Business Oriented Language) is the most widely and extensively used language for the programming of commercial and administrative data processing.

CIS COBOL is a Compact, Interactive and Standard COBOL language system for use on the BASF 7100 microcomputers with CRT and floppy diskettes under control of the BOS Operating System.

The CIS COBOL compilation system converts CIS COBOL source code into an intermediate code which is then interpreted by a Run Time System (RTS).

CIS COBOL programs can be created using the standard BOS text editor to create the CIS COBOL source files. The Compiler compiles the source programs from here, or they are entered interactively direct from the CRT. After compilation is finished, the Run-Time system is linked with the compiled output to form a running user program. A listing of the CIS COBOL program is provided by the Compiler during compilation. Any error messages are included in this listing. Interactive Debugging facilities are provided for runtime use.

The CIS COBOL System also incorporates a powerful utility program called FORMS.

The purpose of FORMS is to allow the user to define the screen layouts to be used in a CIS COBOL application, by simply keying text at the keyboard and so producing model forms on the CRT. The forms can be automatically used to generate a program which will maintain files with the form data in them.

It provides an ideal medium of communication between the programmer and the end user who may know nothing of computers. The minimum storage requirements for FORMS is 64k bytes.

#### GETTING STARTED WITH CIS COBOL

##### ISSUE DISC

Each user is provided with the software that makes up the development system described above on a CIS COBOL Issue Disk

A CIS COBOL Issue Disk contains the software listed in Table 1-1.

Table 1-1. Issue Disk Contents.

| COMPILER                                                  | RUN TIME SYSTEM | DEMONSTRATION PROGRAMS             | RUN TIME SUBROUTINES             |
|-----------------------------------------------------------|-----------------|------------------------------------|----------------------------------|
| COBOL<br>COBOL.I01<br>COBOL.I02<br>COBOL.I03<br>COBOL.I04 | RTS             | PI.CBL<br>STOCK1.CBL<br>STOCK2.CBL | CALL.ASM<br>CALL.HEX<br>CALL.PRN |

If your issue disk does not include these items, refer to your BASF Distributor

#### THE COMPILER

The CIS COBOL Compiler is overlaid and loads its overlays from the drive containing COBOL. The root is contained in COBOL and the overlays are contained in the other COBOL files.

#### THE RUN TIME SYSTEM

The Run Time System (RTS) executes the intermediate code output from the compiler. In addition to standard ANSI COBOL statements, CIS COBOL contains many extensions for use with interactive programs.

#### THE DEMONSTRATION PROGRAMS

PI.CBL, STOCK1.CBL and STOCK2.CBL are simple demonstration programs, supplied in source form, which show many of the facilities present in CIS COBOL, and which can also be used by the newcomer to familiarise himself with the system.

#### THE RUN TIME SUBROUTINES

These modules are supplied to provide an example of the use to which this COBOL CALL facility can be put in implementing RUN TIME Sub-Routines (See Chapter 4). A copy of the list file can be found in Appendix G.

## FIRST STEPS

### Initialisation

Initialise and format system disks as required (see DISK UTILISATION below) and COPY THE CONTENTS of the Issue disk to become a working CIS COBOL system.

### Disk Utilisation

CIS COBOL can be used on a single diskette drive system but is also designed to take full advantage of two-drive systems.

It should be remembered that if only one drive is to be used for compilation and running, a clean system disk with only the BOS system on it should be used with single density disks, to allow sufficient space to receive the CIS COBOL system.

In normal two-drive use, however, it can be beneficial to copy the compiler to one system disk and the Run-Time System (RTS) to another. By default the intermediate code is output to the disk containing the source at compilation and if, therefore, this also contains the RTS, the program can immediately be run. It is the user's responsibility to decide on the most efficient disk allocation for this system.

### Compilation

Compile all the demonstration programs. These have extension .CBL.

#### EXAMPLE:

```
EDIT [ RUN COBOL:0 STOCK1.CBL<<
**CIS COBOL V4.3 COPYRIGHT (C) 1979 MICRO FOCUS LTD
**COMPILING STOCK1.CBL
**ERRORS=00000 DATA=00636 CODE=00222 DICT=00420:nnnnn END OF LIST
```

```
EDIT
```

#### NOTE:

All the examples in this manual assume that the CIS COBOL software diskette is loaded in drive 0. If the diskette was loaded in drive 1, the first line in the above example would be:

```
EDIT [ RUN COBOL:1 STOCK1.CBL<<
```

A directory listing of the disk will show that two new files exist, namely STOCK1.LST which is the list file, and STOCK1.INT which contains the intermediate code. Similar procedures should be followed for STOCK2.CBL and PI.CBL.

Note that STOCK2 has a bug in it which is present to show error Formats and is for demonstration purposes only. It does not affect the running of the program.





written to disk. The unprotected areas will then be space filled ready for the next record to be entered, if the record has been correctly entered. If the record remains displayed, the record was incorrectly keyed.

To terminate the run, enter spaces into the STOCK CODE field and press RETURN.

This results in:

```
END OF PROGRAM
```

Stock Control Program Two (Data Input)

```
EDIT [ RUN RTS:0 STOCK2.INT<<
```

This clears the screen followed by -

```
GOODS INWARD
```

```
STOCK CODE:   < >
ORDER NO     < >
DELIVERY DATE MM/DD/YY
NO OF UNITS  < >
```

This is a skeleton stock data input program. in which the stock records created by STOCK1 can be accessed.

The same cursor control features are present as in STOCK1.INT. Note that the DELIVERY DATE has a different method of prompting than has so far been used.

Terminate in the same way as for STOCK1.

#### PROGRAM DEVELOPMENT CYCLE

The cycle for development and running of CIS COBOL application programs that must be performed by the programmer is as shown in Figure 1-1.

PREPARATION:

The source programs are created on diskette with the user's own existing editor program, or is keyed in directly on the CRT.

COMPILATION:

COBOL PROG.SRC...

... Loads the single pass compiler to convert a source program (PROG.SRC in this example) into an interpreted object form known as Intermediate Code (PROG.INT). The user may specify the file on which the listing will appear. If this is a disk file, it may be edited to correct errors and used as input for the next run of the compiler.

RUNNING:

RUN PROG.INT...

... Loads the Run Time System which in turn loads the Intermediate Code. To aid debugging, the CIS COBOL interactive debugging facility is available. This allows the user to set break points, examine and modify locations and then continue execution.

Once loaded the programs run to process the user files as required by the application and controlled by the Operator through the CRT.

Once the user program is fully tested it may be permanently linked to the Run Time System by use of the "=" option. See Chapter 3.

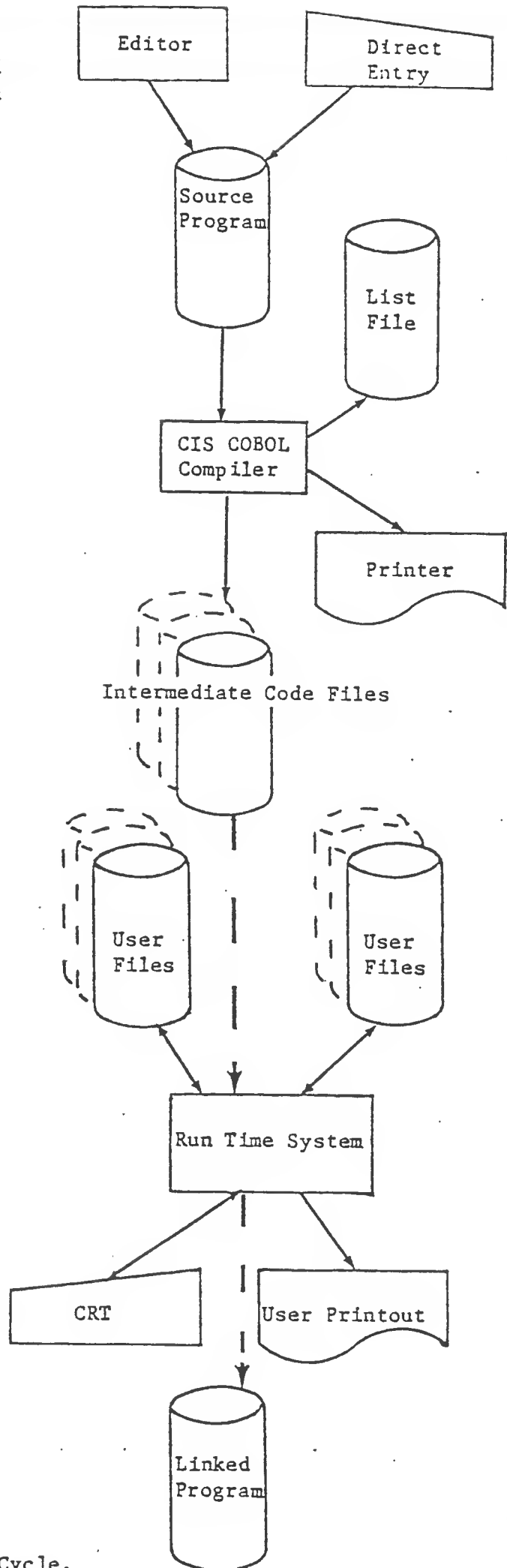


Figure 1 - 1. Program Development Cycle.

## CHAPTER 2

### COMPILER CONTROLS

#### COMMAND LINE SYNTAX

The command line format is:

```
RUN COBOL:Ø filename [directives]<<
```

COBOL is the name of the file which contains the compiler

filename is the optional name of the program which contains the CIS COBOL source statements. If the filename is not given, the console is taken as the input file.

directive is an optional sequence of CIS COBOL directives. Each directive must be separated by one or more spaces. If the sequence is too long to fit on one line of the screen then it may be continued on a subsequent line by typing an ampersand sign "&" followed by carriage return. A particular directive may be on one line only. Where directives have brackets the left-hand bracket may occur zero, one or more spaces after the body of the directive. To terminate the sequence, press return.

#### COMPILER DIRECTIVES

A description of each of the available compiler directives follows:

##### ANS

If this option is specified then the compiler will accept only those CIS COBOL language statements that conform to the ANS74 COBOL standard X.23 1974. The default is "extended" which allows the CIS COBOL extensions and also relaxes the requirement for the COBOL "red tape" statements such as DIVISION headers.

##### RESEQ

If specified, the compiler generates COBOL sequence numbers, re-numbering each line in increments of 10. The default is that sequence numbers are ignored and used for documentation purposes only.

##### NOINT

No intermediate code file is output. The compiler is in effect used for syntax checking only. The default is that intermediate code is output.

##### NOLIST

No list file is produced; used for fast compilation of "clean" programs. The default is a full list.

## COPYLIST

The contents of the file(s) nominated in COPY statements are listed. The list file page headings will contain the name of any COPY file open at the time a page heading is output.

## NOFORM

No form feed or page headings are to be output by the Compiler in the list file. The default is headings are output.

## ERRLIST

The listing is limited to those COBOL lines containing syntax errors together with the associated error message(s). The default is a full list.

## INT (external-file-name)

Specifies the file to which the intermediate code is to be directed. The default is: source-file.INT.

## LIST (external-file-name)

Specifies the file to which the listing is to be directed (this may be a printing device, ie. console or printer or a disk file) The default is: source-file.LST

For list to console use: LIST(:CO:)  
For list to line printer use: LIST(:LP:) (Parallel port)  
LIST(:LS:) (Serial port)  
(See Appendix F for setting up a serial port)

## FORM (integer)

Specifies the number of COBOL lines per page of listing (minimum 5). The default is 60.

## NOECHO

Error lines are echoed on the console unless this directive is specified.

## DATE (string)

The comment-entry in the DATE-COMPILED paragraph, if present in the program undergoing compilation, is replaced in its entirety by the character string as entered between parentheses in the DATE compiler directive.

## EXCLUDED COMBINATIONS

Certain of these directives may not be used in combination. Table 2-1 shows the directives that are excluded if the directive shown adjacent in the left hand column is specified

Table 2-1. Excluded Combinations of Directives

| <u>DIRECTIVE</u> | <u>EXCLUDED DIRECTIVES</u>                             |
|------------------|--------------------------------------------------------|
| NOLIST           | LIST<br>NOFORM<br>FORM<br>RESEQ<br>COPYLIST<br>ERRLIST |
| ERRLIST          | RESEQ<br>COPYLIST                                      |

SUMMARY INFORMATION ON CRT

The general format of the basic command line is:

```
RUN COBOL:Ø filename [directives]<<
```

and the Compiler will reply with:

```
**CIS COBOL Vv.r
```

where v is the version number and r is the release number.

Each directive is then acknowledged by the Compiler on a separate line, and is either ACCEPTED or REJECTED. After all the directives have been acknowledged, the Compiler opens its files and starts to compile. At this point it will display the message:

```
COMPILING filename
```

If any file fails to open correctly, the Compiler will display:

```
filename FAILED TO OPEN
```

The compilation will be aborted, returning control to the operating system.

When the compilation is complete the Compiler displays the message:

```
**ERRORS=nnnnn DATA=nnnnn CODE=nnnnn DICT=mmmmm:nnnnn
```

where:

- ERRORS - denotes the number of errors found
- DATA - denotes the size of RAM required i.e. data area of the generated program
- CODE - denotes the size of ROM required i.e. code area of the generated program
- DICT - denotes the number of bytes used and the number remaining in the data dictionary.

## LISTING FORMATS

The general layout of the list file is as follows:

```
**CIS COBOL Vv.r          filename          PAGE:          nnnn
**
statement 1              HHHH
.
.
.
statement n              HHHH

**CIS COBOL V4.3 COMPILER COPYRIGHT (C) 1978 MICRO FOCUS LTD URN AA/0300/BA
**
**ERRORS=nnnnn DATA=nnnnn CODE=nnnnn DICT=mmmmm:nnnnn          END OF LIST
```

The first two lines of title information are repeated for each page. The final line is the same as on the CRT display. The value denoted by HHHH is a hexadecimal value denoting the address of each dataname or procedure statement. Addresses of datanames are relative to the start of the data area, while addresses of procedure statements are relative to the start of the code area (There is an overhead at the start of the data area, and a few bytes of initialisation code at the start of the procedure area for each SELECT statement).

## CHAPTER 3

### RUN TIME SYSTEM CONTROLS

#### RUN-TIME DIRECTIVES

##### COMMAND LINE SYNTAX

The command line syntax for running a CIS COBOL object program is as follows:

```
RUN [load param] [switch param] [link param] filename [program params]
```

filename is the name of the intermediate code file. File and device conventions for CP/M are given in Appendix F. RUN must have at least one space keyed after it, and filename must have either a space or RETURN keyed after it. The parameters need not have spaces keyed after them. An example of the whole RUN command line is given later in this Chapter.

##### Load Parameters

The optional load parameter provides the Run Time System loader with the load point for the intermediate code in memory. The user has the option to overlay optional modules to conserve program space. Additionally the CIS COBOL Interactive Debug may be invoked. The memory layout of the Run Time System is as shown in Figure 3-1.

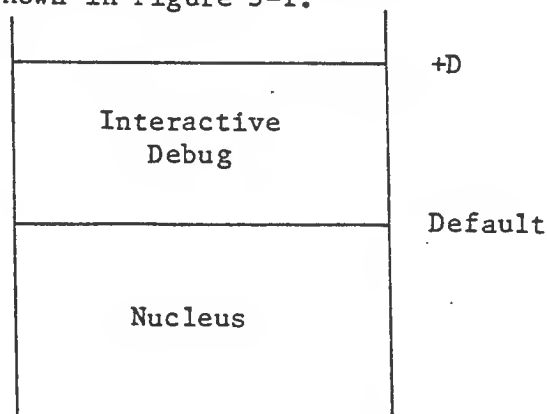


Figure 3-1. Run Time System Memory Layout.

The default load position excludes the Debug module. It may be included by using the parameter "+D". This also implies that Indexed Sequential is included.

If no load parameter is supplied then the intermediate code will be loaded to exclude Debug but retain the I-O packages.

## Switch Parameter

CIS COBOL includes the facility of controlling events in a program at run time depending on whether or not programmable switches are set by the operator. See the description of the SPECIAL-NAMES paragraph in the CIS COBOL Language Reference manual. The operator sets these switches at run time by use of the Switch Parameter to the RUN command. The general format of the Switch Parameter is:

$$\left[ \left( \left\{ \pm \right\}_{n1}^D \left[ [ , ] [ ] \right\} \pm_{n2} \right) \dots \right]$$

where:

- [ ] - denotes an optional item
- { } - denotes a choice
- n1 and n2 - are any numbers in the range 0-7. They can be specified in any order and the last appearance of any specific number takes precedence.
- D - see Standard ANSI COBOL Debug Switch Parameter below
- + or - - set the switch n1, n2, etc. on or off respectively. The default is that all switches are off initially.
- ... - denotes that the preceding options enclosed in the outermost brackets can be repeated.

See EXAMPLES later in this Chapter.

## Standard ANSI COBOL Debug Switch Parameter

Users may also include a parameter to invoke the standard ANSI COBOL Debug module, whether or not the CIS COBOL Interactive Debug extension to ANSI COBOL is invoked. (See the Language Reference Manual for a description of the Debug facilities).

To include the standard ANSI Debug facility a Run Time switch is required. The format is as for a normal switch parameter (see Switch Parameter above), but the numeric switch character is replaced by D. See also EXAMPLES later in this chapter.

## Link Parameter

When the program is fully tested it may be linked with the Run Time System to produce an executable program that can be directly loaded. This is achieved by including the parameter "=" to the Run Time System (see the EXAMPLE overleaf). When the intermediate code file has been loaded (following the lines above) a binary file with the filename SAVE is produced from the current store image. It is essential to rename the SAVE file, from which to load directly, to prevent it being overwritten on the next use of '=' parameter.

See the BOS operating documentation for details of the file renaming command.



## Program Parameters

These are any parameters required by the program, they can be read in on the console file device :CI:.

### COMMAND LINE EXAMPLES

1. The directive

```
RUN RTS:Ø PROG.INT:1 1 2<<
```

loads the program PROG from the intermediate file produced by the compiler and passes the user program parameters 1 and 2 to the program PROG. The I-O packages are included and Debug is omitted.

2. The directive

```
RUN PROG:Ø<<
```

loads the PROG program including the I-O packages but omitting Debug. PROG must have been previously linked by the "=" link parameter.

If it is required to load the sample program STOCK1 in future, instead of the RUN command given in Chapter 1 (A>RUN STOCK1.INT), the following command could be entered:

```
RUN RTS:Ø = STOCK1.INT<<
```

followed by the RENAME command.

In subsequent loads only the command RUN STOCK1:n<< would then be required.

3. The directive

```
RUN RTS:Ø +D (+1+2,+3) = PROG.INT<<
```

loads the program PROG with interactive CIS COBOL Debug and the I-O packages, programmable switches 1, 2 and 3 are set, and a binary file of the program PROG is created, which can subsequently be loaded directly. A SAVE file is created and the interactive CIS COBOL Debug initial display will appear on the CRT when the saved binary PROG is run.

4. The directive

```
RUN RTS:Ø -R (-2 +5-7+7) PROG.INT<<
```

loads the program PROG from the intermediate file produced by the compiler, without any optional packages and with programmable switches 5 and 7 on and 2 off. Note that the last setting of switch 7 is accepted. Switches 1, 3, 4 and 6 are off by default.

5. The directive

```
RUN RTS:Ø (+D) PROG.INT<<
```

loads the program PROG from the intermediate code file produced by the compiler with the standard COBOL ANSI Debug module invoked.

6. The directive

```
RUN RTS:Ø +D (+2,+4 +D) PROG.INT<<
```

loads the program PROG with interactive CIS COBOL Debug and the I-O packages, with programmable switches 2 and 4 set, and with the standard ANSI COBOL Debug module invoked.

## INTERACTION IN APPLICATION PROGRAMS

### CRT SCREEN HANDLING

COBOL is traditionally a batch processing language; CIS COBOL extends the language to make it interactive. CIS COBOL offers many facilities for automatic formatting of a CRT screen and facilitates keying of input.

The CIS COBOL programmer can specify areas of the screen into which the operator is able to key data, and also whether such data is numeric or alphanumeric. This is achieved by defining the screen as a record in the DATA DIVISION in which the data fields correspond to the input area and FILLER's correspond to the rest of the screen.

An ACCEPT statement nominates a record description, which permits input to the character positions corresponding to variables identified by data-names. Conversely, a DISPLAY statement outputs only from non-FILLER fields in the record description which it nominates. The programmer can thus easily build up complex conversations for data entry and transaction processing.

While data is being keyed, the operator has full cursor manipulation facilities, each variable acting as a tab stop. Non-numeric digits may not be entered into fields defined as numeric. Finally, when the operator has checked that the data is correct, the RETURN key is depressed and the data becomes available to the program. Because all characters are transferred to the appropriate area as they are keyed in there is no transmission delay.

### Screen Layout and Format Facilities

The following facilities are available for screen layout and formatting:

- \* Screen as a record description
- \* FILLER
- \* REDEFINES
- \* AT line column

- \* CURSOR addressing
- \* Character highlighting
- \* Clear screen
- \* Numeric validation of PIC 9(n) fields
- \* Automatic editing of numeric edited data-items
- \* De-editing of numeric edited to numeric data-items

Cursor Control Facilities

During execution of ACCEPT statements the cursor is manipulated on the CRT screen by the cursor control keys on the console keyboard as shown in Table 3-1.

Table 3-1. CRT Cursor Control Keys

| Function               | Keys <sup>1</sup>               |
|------------------------|---------------------------------|
| Home                   | HOME                            |
| Tab forward a field    | ↓                               |
| Tab backward a field   | ↑                               |
| Forward Space          | →                               |
| Backward Space         | ←                               |
| Column Tab             | TAB                             |
| Left Zero <sup>2</sup> | .                               |
| Return                 | RETURN<br>or ENTER <sup>3</sup> |

1 - Where CTL is specified the operator must press the CTL key, hold it down and simultaneously press the character key.

2 - The "." for left zero fill is a "," when  
DECIMAL-POINT IS COMMA  
is specified in the user program

3 - Note that when using Debug commands, << represents pressing the Enter key (without End of Message) or the Return Key.

## INTERACTIVE DEBUGGING

Two levels of debugging are available to the programmer. The first involves optional "debugging lines" that are included if the "DEBUGGING MODE" switch is present in the "SOURCE-COMPUTER" sentence. The second is the interactive Debug package that is included at run-time under the control of the user (see section on Run Time System).

If Debug is included in the Run Time System, it will announce its presence when the program is loaded as follows:

```
RUN RTS:Ø +D STOCK1.INT<<
Debug Mark 3.1           -title
>                        -prompt
```

The user now has the following commands available:

- P - Displays the current program counter (p-c).
- G - Breakpoint at specified address.
- X - Execute one CIS COBOL statement at a time.
- D - Display bytes in the Data Division
- A - Replace contents of a memory location by a hexadecimal value or ASCII character.
- S - Set start of block for correction or display.
- / - Display bytes in block above.
- . - Change bytes in block above.
- T - Trace paragraphs up to breakpoint specified.
- L - Output one CR LF on the CRT
- M - Define Debug command macro with name specified
- \$ - End macro definition
- C - Displays specified character on the CRT
- ; - Precedes comment to describe a macro just entered.

A description of the use of each of these Debug commands follows.

### THE P COMMAND

The P command displays the address at which the program counter currently points i.e. where the current instruction is in the Procedure Division code of a program. This hexadecimal address is that printed in the right hand column of a program existing.



made up of several individual primitive instructions, DEBUG may appear to halt in the middle of a line. If this occurs, the RETURN is pressed again.

EXAMPLE:

If an error occurred in the MOVE instruction the X command sequence would be shown as follows:

```
>X<<
 018C
>
```

To check the contents of "FIELD-2" before and after the move for code in the "DATA DIVISION" the display would be:

```
02 FIELD-1 PIC XXX VALUE "ABC".      0030
02 FIELD-2 PIC XXX VALUE "XYZ".      0033
02 FIELD-3 PIC X(80) VALUE SPACE.    0036
.
.
```

To display bytes in the DATA DIVISION, the 'D' command can be used. This displays 16 bytes from the address specified (again the address is derived from the information on the listing). It displays each byte as a hexadecimal value plus an ASCII equivalent if it is printable.

EXAMPLE:

```
>D 0030<<
41-A 42-B 43-C 58-X 59-Y 5A-Z 20- 20- 20- .....
FIELD-1      FIELD-2      FIELD-3
>
```

If the MOVE and is then executed and re-examined the following display results:

```
>X<<
 0190
>D 0030<<
41-A 42-B 43-C 41-A 42-B 43-C 20- 20- 20- .....
```

### THE A COMMAND

The "A" command is used to amend data at a specified memory location.

EXAMPLE:

To replace the first character "A" of FIELD-1 by "G". The value supplied may be a two character hex value or an ASCII character preceded by quote eg "G" or 47.

```
>A 0030 47<<          -amend byte
>D 0030<<
 47-G 42-B 43-C 41-A 42-B 43-C 20- 20- 20- .....
>
```

This correction facility allows continued running even if a bug has produced an erroneous result.

#### THE S COMMAND

Where a number of corrections are required, DEBUG allows specification of a working register which contains an address. This address can be set or incremented and the contents can be displayed or modified immediately by use of the 'S' command. The address and contents can then be displayed by keying '/'.

#### EXAMPLE:

To display the first byte of FIELD-1 operation would be as follows:

```
>S 0030<<          -load address
>/<<              -display
  0030 47G
>
```

#### THE '.' COMMAND

To amend the byte at the current location '.' is used; this also increments the working register.

#### EXAMPLE:

To change FIELD1 to "DEF" the display would be:

```
>S 0030<<          -load address
>.44.45.46<<      -modify
>D 0030<<
  44-D 45-E 36-F .....
>
```

To increment only the working register use ','.

#### THE T COMMAND

An advanced form of the 'G' command is the 'T' command. This also executes up to a breakpoint in the PROCEDURE DIVISION, but also prints the address of each paragraph encountered.

#### EXAMPLE:

```
>T 017B<<          trace up to 017B
```

## DEBUG MACRO COMMANDS

The user will find that some Debug command sequences are used often when debugging. If these sequences are long it can become tiresome typing them in. To overcome this and to allow the development of complex debugging sequences Debug permits the definition of macros comprised both of basic operations and other macros. Macros are given names of one character.

### The M Command

Macros are introduced by the 'M' command followed immediately by the macro name.

#### EXAMPLE:

To define a macro to execute up to 018C, display the value at 0030, then single-step and display again, the following would be typed:

```
>MZ G 018C  D0030 L X D 0030 $<<
>
To invoke this macro its name is typed as follows:

>Z<<
41-A 42-B 43-C 58-X 59-Y 5A-Z .....   First display
0190
41-A 42-B 43-C 41-A 42-B 43-C .....   Second display
>
```

There are two other commands introduced in this macro: 'L' and '\$'.

### The L Command

The 'L' command merely forces a carriage return and line feed to be output on the console.

### The \$ Command

The '\$' command ends a macro definition.

### The C Command

To allow macro writers to output characters to the console, the command 'C' is provided. This outputs its parameter on the console

#### EXAMPLE:

```
>C "A<<
A
>
```

### The ; Command

To improve readability of macros, comments may be inserted. These are introduced by the character ';' and terminated by carriage return.



EXAMPLE:

```
>MZ D 0030 X L D 0030 $ ; Run macro<<
```

Macro names must be letters only. Lower case letters are converted internally to upper case.

If an error is made in typing in a macro then it may be reentered. However, there is only a finite amount of macro space and space is not released if a macro is reentered. If the space runs out or the maximum nesting of macros is exceeded then the message STACK OVERFLOW will result.

EXAMPLE:

```
>MZ Z$ ; macro to crash system<<  
>Z<<
```

After the crash has occurred, the Debug system will return to command mode and will generally tidy up the stack to allow the user to continue. However, if more serious crashes occur i.e. those with no message, then the system will not recover.

For full details of Debug commands see Appendix E.

## CHAPTER 4

### MULTILANGUAGE CALL FACILITIES

#### INTRODUCTION

CIS COBOL enables other COBOL programs to be called from a main application program, and also enables programs written in other languages to be called from a main COBOL application program.

#### COBOL PROGRAMS - CALL

##### THE RUN UNIT

An application written in COBOL may be arranged into a number of separate COBOL programs which communicate and invoke each other by use of the COBOL "CALL" verb.

##### FORMAT OF CIS COBOL "CALL"

The general format of the CIS COBOL verb is given in the CIS COBOL Language Reference Manual.

##### FORM OF CIS COBOL PROGRAMS

Each program in an CIS COBOL application suite must be written in COBOL and with the exception of the main program should have a Linkage Section in the Data Division with which to communicate with other COBOL programs.

Any COBOL program other than the main program must be compiled and its intermediate code placed on a disk which is accessed at run time. The main program may be in intermediate code and named as a parameter to RUN, or it may be linked to RUN in the manner described earlier under Run-Time Directives.

##### RUN TIME PROGRAM LINKAGE

Run time execution of the COBOL verb CALL depends on the parameter used by the CALL. COBOL programs and assembler code subroutines can be CALLED.

When the parameter is an alphanumeric quantity its value is interpreted as a file-name and the appropriate file of intermediate code is loaded from disk into memory and executed.

When the parameter is a numeric quantity, its value is interpreted as the Linkage number to the Run-Time Subroutine Table and the corresponding machine code subroutine is executed. The subroutine must be configured into the RTS for the main program (See RUN-TIME SUBROUTINES - CALL in this Chapter).

## EXAMPLE LINKAGE

PROCEDURE DIVISION

CALL "SUBITM.INT:Ø" USING ...

CALL "10" USING ...

For the first CALL in this example to perform correctly the file SUBITM.INT must be present on disk unit Ø and must contain a compiled COBOL program. For the second CALL to perform correctly the RTS must contain a machine code subroutine arranged as subroutine 10.

## LIMITATIONS OF CALL

Any number of COBOL programs and assembler code subroutines can be CALLED from a COBOL program. Operational limitations on CALL are as follows:

1. The CALLED intermediate code program file must be present on disk at the time of the CALL.

It is possible to load the program into memory for the program to be loaded.

The program must be pre-configured into the RTS.

It should be noted by noting that:

The program can be loaded at run time by suitable user-programmed subroutines.

It is possible to check whether a CALL has failed due to lack of

available storage when executed at run time.

In large and complex CIS COBOL applications, application designers in particular should realise that the limitation is not constrained by the intrinsic

## RUN TIME SUBROUTINES - CALL

### USER SUBROUTINES

The Run Time System is designed in such a way that the user may write and include Assembler or other language subroutines that can be accessed using the COBOL "CALL" verb. (See Appendix G for example of use of this facility). If you require to write your own non-COBOL subroutines contact your BASF representative for further details.

### ASSEMBLER SUBROUTINES PROVIDED BY MICRO FOCUS

The following standard CALL codes are available in the Run Time System.

|        |   |                 |
|--------|---|-----------------|
| CHAIN  | - | CALL code "260" |
| PEEK   | - | CALL code "261" |
| POKE   | - | CALL code "262" |
| GET    | - | CALL code "263" |
| PUT    | - | CALL code "264" |
| ABSCAL | - | CALL code "265" |

The user may call these routines without making any alteration to the Run Time System.

## The CHAIN Subroutine

The CHAIN call allows another linked CIS COBOL program or any program not requiring parameters to be loaded and entered. There is no return to the calling program.

A parameter list of one variable must be passed with CALL CHAIN:

- \* The data-name containing the file-name of the program to chain to. The file-name must be terminated by at least one space character.

### EXAMPLE:

WORKING-STORAGE SECTION.

```
      .  
      .  
      .  
03    NEXT-PROG PIC X(10) VALUE "PRIN2.OBJ "  
      .  
      .  
03    CHAIN     PIC X(3)  VALUE "260".
```

PROCEDURE DIVISION.

```
      .  
      .  
      .  
      CALL CHAIN USING NEXT-PROG.  
      .  
      .  
      .
```

Ná RTS:0 = PRØG.INT:1

Next-prog is dow " PRØG:1 Δ".

## The PEEK Subroutine

The PEEK call allows an absolute address location to be examined from a user program. The CALL returns into the user area a copy of the 8 bit value at the absolute address.

A parameter list of two variables must be passed with CALL PEEK:

- \* The five character data-name containing the absolute address to be read from.
- \* The one character data-name where the value is to be read to.

EXAMPLE:

WORKING-STORAGE SECTION.

```
      .  
      .  
      .  
03    PEEK    PIC X(3) VALUE "261".  
      .  
      .  
      .  
03    ADDRESS PIC 9(5) VALUE 1234 .  
      .  
      .  
      .  
03    DATA-VAL PIC X.  
      .  
      .  
      .  
      .
```

PROCEDURE DIVISION.

```
      .  
      .  
      .  
      .  
      CALL PEEK USING ADDRESS, DATA-VAL.  
      .  
      .  
      .
```



## The PUT Subroutine

The PUT call allows a hardware port to be output from a user program. The CALL outputs an 8 bit value to the port from a user area.

A parameter list of two variables must be passed with CALL PUT:

- \* The three character data-name containing the port to be written to.
- \* The one character data-name to be written from.

EXAMPLE:

WORKING-STORAGE SECTION.

```
      .  
      .  
03   PUT      PIC X(3) VALUE "264".  
      .  
      .  
03   PORT     PIC 9(3) VALUE 131.  
      .  
      .  
03   DATA-VAL PIC X      VALUE X"2F".  
      .  
      .
```

PROCEDURE DIVISION.

```
      .  
      .  
      CALL PUT USING PORT, DATA-VAL.  
      .  
      .  
      .
```



The ABSCAL Subroutine

The ABSCAL call allows a subroutine CALL to an absolute location. No parameters are passed to the subroutine at the absolute address.

A parameter list of one variable must be passed with CALL ABSCAL:

- \* The five character data-name containing the absolute address to be called.

EXAMPLE:

WORKING-STORAGE SECTION.

03 ABSCAL PIC X(3) VALUE "265".

03 ADDRESS PIC 9(5) VALUE 5.

PROCEDURE DIVISION.

CALL ABSCAL USING ADDRESS.

## the BSFSUB1 Subroutine

This subroutine allows execution of FCS-commands or MTX80 from a COBOL program, where the function of this routine depends on the value of the first parameter.

parameter list of four variables has to be passed with CALL BSFSUB1:

If a FCS-command (usually STATUS, COPYDISK, COPYFILE, CHANGEFILE, ERASE, ATTRIB, REORG OR EXECUTE PGM) is to execute, the list of parameters include:

- \* a one character routine identifier valued "1".
- \* a max. 80 character command field containing the FCS-command as described in FCS manual Appendix II: ~filename:unit # ~opcode [parameters] and HEX "FF" to indicate the end of command string.
- \* a two character return-code to receive a FCS error code (in form of XX for 7XX).
- \* the result field which depends in its length on the special command.

## EXAMPLE:

## WORKING-STORAGE SECTION.

\*..... PRINT STATUS:1.....

|    |                 |             |              |
|----|-----------------|-------------|--------------|
| 01 | FCS-COMMAND.    |             |              |
| 02 | FILLER          | PIC X       | VALUE "~".   |
| 02 | FILENAME        | PIC X(16)   | VALUE SPACE. |
| 02 | FILLER          | PIC X       | VALUE ":".   |
| 02 | DRIVE           | PIC 9       | VALUE "1".   |
| 02 | FILLER          | PIC X       | VALUE "~".   |
| 02 | OPCODE          | PIC X       | VALUE "?".   |
| 02 | PARAMETER       | PIC X(34)   | VALUE SPACE. |
| 02 | COMMAND-END     | PIC XX      | VALUE X"FF". |
|    |                 |             |              |
| 01 | BSFSUB1         | PIC XX      | VALUE "00".  |
|    |                 |             |              |
| 01 | CALL-PARAMETER. |             |              |
| 02 | FCS-CALL        | PIC X       | VALUE "1".   |
| 02 | RETURN-CODE     | PIC 99.     |              |
| 02 | RESULT          | PIC X(400). |              |

## PROCEDURE DIVISION.

CALL BSFSUB1 USING FCS-CALL, FCS-COMMAND, RETURN-CODE, RESULT.  
IF RETURN-CODE NOT = ZERO .....

the BSFSUB1 Subroutine

The BSFSUB1 call is used to execute MTX80 function (e.g. SET TIME OR GET TIME) the list of four parameters include

- \* a one character routine identifier valued "2".
- \* a variable character field as normally passed to MTX80 from BASIC as described in MTX/80 manual, which contains a one character function code optionally followed by a slash and a parameter string and HEX "FF" to indicate the end of command string.
- \* a two character return-code as passed back from MTX80.
- \* the result field which contains the result (if applicable).

EXAMPLE:

WORKING-STORAGE SECTION.

```

*..... SET DATE.....

01 MTX-COMMAND.
   02 MTX-CODE          PIC X      VALUE "C".
   02 FILLER           PIC X      VALUE "/".
   02 MTX-PARM         PIC X(17)  VALUE "MM/DD/YY HH:MM:SS".
   02 COMMAND-END     PIC XX     VALUE X"FF".

01 BSFSUB1            PIC XX     VALUE "00".

01 CALL-PARAMETER.
   02 MTX-CALL         PIC X      VALUE "2".
   02 RETURN-CODE     PIC X(2)   VALUE XX.
   02 RESULT          PIC X(400).
    
```

PROCEDURE DIVISION.

```

CALL BSFSUB1 USING MTX-CALL, MTX-COMMAND, RETURN-CODE, RESULT.
IF RETURN-CODE NOT = ZERO .....
    
```

NOTE : for both commands a HEX "FF" as last byte of second parameter is mandatory.

## CHAPTER 5

### INCORPORATING FORMS UTILITY PROGRAM OUTPUT

#### INTRODUCTION

The FORMS Utility program offers two major facilities to CIS COBOL users:

1. The user can define screen layouts to be used in a CIS COBOL application by simply keying the text at the keyboard, and so producing a model form on the CRT.
2. The user can automatically generate programs to manipulate data input using the created form. In particular, indexed sequential files can be generated and maintained automatically, and these files can, of course, be used with CIS COBOL programs.

The FORMS Utility is available as a separate software package, and is supported by the FORMS Utility Program Users Guide.

#### SCREEN LAYOUT FACILITY

The FORMS Screen Layout facility generates source COBOL Record Descriptions for screen layouts.

#### MAJOR FACILITIES

Users have three major facilities available to them:

1. They may store an image copy on disk of the form they have just defined for subsequent use in this or another FORMS run. The image can be printed to obtain a hard copy, using the O/S standard file print utility program.
2. They may generate CIS COBOL source code for the data descriptions required to define the form just created. This may then be included into a CIS COBOL program by use of the COPY verb.
3. They may choose to generate a Check Out program which allows duplication of many machine conversations which would take place during a run of the application which is being designed.

#### CIS COBOL PROGRAMMING FOR FORMS SCREEN LAYOUTS

All that the user has to do to incorporate FORMS Screen layout output in a program is to specify the FORMS output file name (filename.DDS) in a COBOL COPY statement. Obviously data item names in the user program must be specified to correspond with those generated from a user-specified base name by FORMS. Details of FORMS name generation are given in the FORMS Utility Program Users Guide.

#### EXAMPLE:

```
000000 COPY "DEMO.DDS".
```

## GENERATED PROGRAMS

The FORMS Utility generates a COBOL program which maintains data stored in the created forms in an indexed sequential file automatically, with automatic generation of file names from a user-supplied base name. These files comply with the standards in use by the operating system under which CIS COBOL is being used.

### CIS COBOL PROGRAMMING FOR FORMS GENERATED FILES

No special programming is required to use FORMS generated program files in a CIS COBOL application program. The files are processed as normal indexed sequential files. It is worth noting that the files can be fully maintained interactively by use of only the FORMS Utility. In addition to establishing or deleting files, this includes the following facilities:

- \* Insertion of new records
- \* Insertion of the same data in records with different keys
- \* Display of any selected record/s (Full inquiry facility)
- \* Insertion or amendment of records dependent on their key
- \* Deletion of records
- \* Read and display next record or a message if end of file detected
- \* Terminate run

Details of the FORMS Indexed Sequential File handling facilities are given in the FORMS Utility Program Users Guide.

## APPENDIX A

### SUMMARY OF COMPILER AND RUN TIME DIRECTIVES

#### COMPILER DIRECTIVES

The general format of the command line for compilation is:

```
RUN COBOL:Ø filename [directives]
```

filename is the name of the file that contains the CIS COBOL source program.

A description of the available compiler directives follows:

#### ANS

If this option is specified, the compiler will accept only those CIS COBOL language statements that conform to the ANS 74 standard. The default is "extended", which allows the CIS COBOL extensions and also relaxes the requirement for the COBOL "red tape" statements such as DIVISION headers.

#### RESEQ

If specified, the compiler generates COBOL sequence numbers, re-numbering each line in increments of 10. The default is that sequence numbers are ignored and used for documentation purposes only.

#### NOINT

No intermediate code file is output. The compiler is, in effect, used for syntax checking only. The default is that intermediate code is output.

#### NOLIST

No list file is produced; used for fast compilation of "clean" programs. The default is a full list.

#### COPYLIST

The contents of the file(s) nominated in COPY statements are listed. The list file page headings will contain the name of any COPY file open at the time a page heading is output.

#### NOFORM

No form feed or page headings are to be output by the compiler in the list file. The default is headings are output.

#### ERRLIST

The listing is limited to those COBOL lines containing syntax errors together with the associated error message(s). The default is a full list.

INT (external-file-name)

Specifies the file to which the intermediate code is to be directed.  
The default is: source-file.INT.

LIST (external-file-name)

Specifies the file to which the listing is to be directed. (This may be a printing device, i.e. console or printer or a disk file.) The default is: source-file.LST.

For list to console use: LIST (:CO:)  
For list to line printer use: LIST (:LP:) (Parallel port)  
LIST (:LS:) (Serial port)

FORM (integer)

Specifies the number of COBOL lines per page of listing (minimum 5).  
The default is 60.

NOECHO

Error lines are echoed on the console unless this directive is specified.

DATE (string)

The comment-entry in the DATE-COMPILED paragraph, if present in the program undergoing compilation, is replaced in its entirety by the character string as entered between parentheses in the DATE compiler directive.

#### RUN TIME DIRECTIVES

The command line syntax for running a CIS COBOL object program is as follows:

```
RUN RTS:n [load param] [switch param] [link param] filename [program  
params]
```

where:

n is a drive number Ø, 1 or 2

load param is one of the following:

Default (Parameter omitted)

Loads the Indexed file handling facilities and the basic Run Time System

+D Loads the CIS COBOL Interactive Debug extension, the Indexed Sequential file handling facilities and the basic Run Time System

switch param is of general format:

$$\left[ \left( \{ \pm \} \left\{ \begin{array}{c} D \\ n1 \end{array} \right\} \left[ [ , ] [ ] \{ \pm \} n2 \right] \dots \right) \right]$$

n1 and n2 are any program switch numbers (See Language Reference Manual) in the range 0-7

D invokes the standard ANSI COBOL Debug module

+ or - sets the associated switch on or off

link param is the = (equal sign) symbol which is used to link the program with the Run Time System so that it can be directly loaded. Note that it is important to rename the SAVE file generated to avoid it being overwritten at the next use of the = parameter

filename is the name of the file in which the intermediate code of the program to be loaded is stored

program params are any formats required to be passed to the program from the Operator at load time. These are user specific.



THIS  
PAGE  
INTENTIONALLY  
BLANK

APPENDIX B

COMPILE-TIME ERRORS

The error descriptions that correspond to error numbers as printed on listings produced by the CIS COBOL compiler are as follows:

| ERROR | DESCRIPTION                                       |
|-------|---------------------------------------------------|
| 01    | Compiler Error                                    |
| 02    | Bad lexical item: data-name                       |
| 03    | Bad lexical item: literal                         |
| 04    | Bad lexical item: character                       |
| 05    | data-name declared twice                          |
| 06    | Dictionary overflow                               |
| 07    | Illegal character in column 7                     |
| 08    | Compiler I-O failure                              |
| 09    | CIS COBOL extension used with ANS directive       |
| 10    | Wrong area A/B                                    |
| 21    | '.' missing                                       |
| 22    | 'DIVISION' missing                                |
| 23    | 'SECTION' missing                                 |
| 24    | 'IDENTIFICATION' missing                          |
| 25    | 'PROGRAM-ID' missing                              |
| 26    | 'AUTHOR' missing                                  |
| 27    | 'INSTALLATION' missing                            |
| 28    | 'DATE-WRITTEN' missing                            |
| 29    | 'SECURITY' missing                                |
| 30    | 'ENVIRONMENT' missing                             |
| 31    | 'CONFIGURATION' missing                           |
| 32    | 'SOURCE-COMPUTER' missing                         |
| 33    | MEMORY SIZE/COLLATING SEQUENCE in error           |
| 34    | 'OBJECT-COMPUTER' missing                         |
| 36    | 'SPECIAL-NAMES' missing                           |
| 37    | SWITCH Clause in error                            |
| 38    | DECIMAL-POINT Clause in error                     |
| 39    | CONSOLE Clause in error                           |
| 40    | Illegal currency symbol                           |
| 41    | '.' missing                                       |
| 42    | 'DIVISION' missing                                |
| 43    | 'SECTION' missing                                 |
| 44    | 'INPUT-OUTPUT' missing                            |
| 45    | 'FILE-CONTROL' missing                            |
| 46    | 'ASSIGN' missing                                  |
| 47    | 'SEQUENTIAL' or 'INDEXED' or 'RELATIVE' missing   |
| 48    | 'ACCESS' missing on indexed/relative file         |
| 49    | 'SEQUENTIAL/DYNAMIC' missing                      |
| 50    | Illegal combination ORGANIZATION/ACCESS/KEY       |
| 51    | SELECT Clause phrase unrecognised                 |
| 52    | RERUN Clause syntax error                         |
| 53    | SAME AREA Clause syntax error                     |
| 54    | file-name missing                                 |
| 55    | 'DATA DIVISION' missing                           |
| 56    | 'PROCEDURE DIVISION' missing or unknown statement |
| 61    | '.' missing                                       |
| 62    | 'DIVISION' missing                                |

63 'SECTION' missing  
64 file-name is not selected  
65 Record size integer missing  
66 Illegal level number (01-49) or 01 level required  
67 FD qualification contains syntax error  
68 'WORKING-STORAGE' missing  
69 'PROCEDURE DIVISION' missing or unknown statement  
70 Data Description Qualifier or '.' missing  
71 SIGN/USAGE illegal with COMP data-item or unsigned  
PICTURE data or incompatible with other qualifier  
72 BLANK is illegal with non-numeric data-item  
73 PICTURE clause too long (Numeric 18 Numeric  
Edited 512 Alphanumeric 8192)  
74 VALUE clause on non-elementary data-item, or truncation,  
or wrong data type  
75 'VALUE' in error or illegal for PICTURE type  
76 FILLER/SYNCHRONIZED/JUSTIFIED/BLANK non-elementary  
item  
77 Level 0 or level with more than 8192 bytes  
78 REDEFINES of unequal fields or different levels.  
79 Data storage exceeds 64K bytes  
80 'DYNAMIC' only allowed in non-ANS and at level 01  
81 Data Description Qualifier inappropriate or repeated  
82 REDEFINES data-name not declared  
83 USAGE must be COMP, DISPLAY or INDEX  
84 SIGN must be LEADING or TRAILING  
85 SYNCHRONIZED must be LEFT or RIGHT  
86 JUSTIFIED must be RIGHT  
87 BLANK must be ZERO  
88 OCCURS must be numeric, non-zero and unsigned  
89 VALUE must be a literal, numeric literal or  
figurative constant  
90 PICTURE string has illegal precedence or illegal  
character  
91 INDEXED data-name missing or already declared  
92 numeric edited PICTURE string is too large  
101 Unrecognised verb  
102 If ... else mismatch  
103 Wrong data-type  
104 Paragraph name declared twice  
105 Paragraph name same as data-name  
106 Name required  
107 Wrong combination of data types  
108 Conditional imperative statement  
109 Malformed subscript  
110 ACCEPT/DISPLAY wrong  
111 Bad I/O Syntax  
116 Ifs nested too deep  
117 Bad skeletal structure of Procedure Division  
118 Obligatory Reserved Word missing  
119 Subscript vector overflow  
120 Intermediate code output buffer overflow  
140 Inter-segment procedure name check  
142 If ... mismatch at end of Source Input  
143 Wrong data-type  
144 Paragraph name undeclared  
145 Index-name declared twice

146 Bad cursor control  
147 KEY declaration missing  
148 STATUS declaration missing  
149 Bad STATUS record  
151 PROCEDURE DIVISION in error  
152 USING parameter not declared in linkage section  
153 USING parameter is not level 01 or 77  
156 I-O Error on auxiliary segmentation files  
157 Bad skeletal structure of Procedure Division  
154 USING parameter used twice in parameter list  
160 Intermediate Code Output buffer overflow

## RUN TIME ERRORS

Run-Time error messages are preceded by the name and segment number of the currently executing intermediate code file.

There are two types of Run-Time errors: Recoverable and Fatal.

## (a) Recoverable errors

If the programmer has selected STATUS for a file then error handling is his responsibility. This will generally only apply to errors that are not considered fatal by the operating system.

## (b) Fatal errors

All errors except those above are fatal. They may arise from the operating system or from the Run Time System. Fatal errors cause a message to be output to the console which includes a 3 digit error code and reference to the COBOL statement in which it occurred. These fall into two classes:

## (i) Exceptions

These cover arithmetic overflow, subscript out of range, too many levels of perform nesting.

## (ii) I-O errors

These exclude those for which STATUS is not selected as above.

| <u>Error</u> | <u>Description</u>                                    |
|--------------|-------------------------------------------------------|
| 151          | Random read on sequential file                        |
| 152          | REWRITE or DELETE on file not open I-O                |
| 153          | Subscript out of range                                |
| 154          | Perform nesting exceeds 22 levels                     |
| 155          | Illegal command line                                  |
| 156          | Invalid file operation                                |
| 157          | Object file too large                                 |
| 158          | REWRITE on line-sequential file                       |
| 159          | Malformed line-sequential file                        |
| 160          | Overlay loading or chaining error                     |
| 161          | Illegal intermediate code                             |
| 162          | Arithmetic overflow or underflow                      |
| 164          | Specified CALL code not supplied                      |
| 165          | Incompatible releases of compiler and Run Time System |

APPENDIX D

OPERATING SYSTEM ERRORS

These errors appear in the same format as the CIS COBOL errors; conventionally error numbers 1-99 are reserved for the operating system.

| <u>Error</u> | <u>Description</u>                            |
|--------------|-----------------------------------------------|
| 4            | Illegal file name                             |
| 5            | Illegal device specification                  |
| 6            | Attempt to write to input file                |
| 7            | Insufficient diskette space                   |
| 8            | Attempt to read from file opened for output   |
| 11           | REWRITE or DELETE sequence error              |
| 12           | Attempt to open file already open             |
| 13           | Attempt to open for input a non-existent file |
| 14           | Protection error                              |
| 24           | Disk system error                             |
| 30           | Drive not ready                               |
| 33           | Bad parameter to file system                  |
| 38           | File full                                     |

APPENDIX E  
INTERACTIVE DEBUG COMMAND SUMMARY

| <u>COMMAND</u> | <u>EFFECT</u>                                                            |
|----------------|--------------------------------------------------------------------------|
| A data-ref val | Change value at address given to val<br>(data division)                  |
| C val          | Display ASCII character corresponding<br>to val                          |
| D data-ref     | Display 16 bytes from address given                                      |
| G proc-ref     | Execute from current position until<br>given address is reached          |
| L              | Output carriage return/line feed to<br>console                           |
| M name         | Start definition of macro                                                |
| P              | Display current program counter                                          |
| S data-ref     | Set work register to address given                                       |
| T proc-ref     | Trace all paragraphs executed up to<br>address (procedure division)      |
| X              | Execute one instruction                                                  |
| \$             | End macro definition                                                     |
| /              | Display byte at address in work register                                 |
| . val          | Change byte at address in work register<br>to val and increment register |
| ,              | Increment work register                                                  |
| ;              | Start comment - line up to carriage<br>return is ignored                 |

where:

|          |   |                                                                          |
|----------|---|--------------------------------------------------------------------------|
| data-ref | - | 16 bit hex value (4 digits)<br>in data area                              |
| proc-ref | - | 16 bit hex value (4 digits)<br>in code area                              |
| val      | - | 8 bit value (2hex digits or<br>inverted commas and ASCII<br>char eg "A") |
| name     | - | single ASCII character                                                   |

## APPENDIX F

### BOS DISK FILES

#### GENERAL

The disk file system used in CIS COBOL is the BOS file control system described in the BOS Users Manual. A description of file creation and management is available in that manual.

CIS COBOL offers sequential, relative and indexed organizations.

All file processing information is defined within an interactive CIS COBOL program. File organization, access method, device assignment and allocation of disk space are defined by the SELECT statement in the INPUT-OUTPUT SECTION of the ENVIRONMENT DIVISION and an FD entry in the FILE SECTION of the DATA DIVISION.

#### SPECIFYING FILES

CIS COBOL offers fixed (compile time) file assignment and dynamic (run time) file assignment facilities.

#### FIXED FILE ASSIGNMENT

The CP/M file name is assigned to the internal user file-name at compile time as shown in the specifications that follow.



Environment Division

In the FILE-CONTROL paragraph the general format of the SELECT and ASSIGN TO statements is as follows:

General Format

SELECT file-name

ASSIGN TO external-file-name-literal  
file-identifier

[ { external-file-name-literal }  
, { file-identifier } ] .

Parameters

- filename - Can be any user-defined  
CIS COBOL word (see User  
Defined COBOL Words in Chapter  
2)
- external-file-name-literal - Is a standard CP/M file name of  
the following general format:

filename . extension : device

- ↑  
a BOS drive number preceded by a colon.  
F0 through F3 - Diskette drives  
(Equivalent to  
Ø: through 2:)
- LP - Line printer  
(Parallel port)
- LS - Line printer  
(Serial port)
- CI - Console input
- CO - Console output

1-3 alphanumeric characters

1-6 alphanumeric characters

- File-identifier - See Run Time file Assignment  
later in this Appendix

Examples of Fixed File Assignment

SELECT STOCKFILE

ASSIGN TO "WAREHS.BUY:1".

SELECT STOCKFILE

ASSIGN TO ":F1:WAREHS.BUY".

## Data Division

The file-name specified as above is then used in the File Description for that program (see The File Description - Complete Entry Skeleton in Chapters 5, 6 and 7 of the CIS COBOL Language Reference Manual)

## Procedure Division

The file-name specified as above is then also used in the OPEN and CLOSE statements when the file is required for use in the program. (See THE OPEN STATEMENT and THE CLOSE STATEMENT in Chapters 5, 6 and 7 of the CIS COBOL Language Reference Manual).

## RUN-TIME FILE ASSIGNMENT

The internal user file-name is assigned to a file-identifier (an alphanumeric user-defined COBOL Word), which automatically sets up a PIC X(25) data area in which to store the external BOS file name. The external BOS file name can then be stored in this data area in the Procedure Division by the user, and can be altered during the run as required.

The following specifications are required for run-time assignment:

### Environment Division

In the FILE-CONTROL paragraph the general format of the SELECT and ASSIGN TO statements is as follows:

#### General Format

```
    SELECT filename  
    ASSIGN TO fileidentifier
```

#### Parameters

|                 |   |                                                                                                                                 |
|-----------------|---|---------------------------------------------------------------------------------------------------------------------------------|
| file-name       | - | Can be any user-defined CIS COBOL word. (See User defined COBOL Words in Chapter 2 of the CIS COBOL Language Reference Manual). |
| file-identifier | - | Is any user-defined CIS COBOL word (See User Defined COBOL Words in Chapter 2 of the CIS COBOL Language Reference Manual).      |

#### Example of Run-Time File Assignment

```
    SELECT STOCKFILE  
    ASSIGN STOCKNAME.
```

## Data Division

The file-name specified as above is then used in the File Description for that program (see THE FILE DESCRIPTION - COMPLETE ENTRY SKELETON in Chapters 5, 6 and 7 of the CIS COBOL Language Reference Manual).

## Procedure Division

The external O/S file name of the required file (see under FIXED FILE ASSIGNMENT above for format) is then stored as required in the file-identifier location specified above by the user program before the file is OPENed for use.

### EXAMPLE:

```
      MOVE      "WAREHS.BUY:1" TO STOCK-NAME.  
      OPEN      INPUT      STOCK-FILE.  
      .  
      .  
      .  
      CLOSE     STOCK-FILE.  
      .  
      .  
      .  
      MOVE      "WAREHS.SEL:1" TO STOCK-NAME.  
      OPEN      INPUT      STOCK-FILE.  
      .  
      .  
      .  
      CLOSE     STOCK-FILE.  
      .  
      .  
      .  
      MOVE      "PROGA.SRC:1" TO file-identifier.  
      OPEN      INPUT      file-name.
```

The BOS file name could have been entered via an ACCEPT statement i.e. by an operator, or stored as any other variable data.

In this way different external files can be used as a common internal user file during any run of a program, but care is required to ensure that the correct file is allocated at any given time.

### NOTE:

The device assignment :1 in the file name above can be replaced by the format :F1: for compatibility with other operating systems.

## SETTING BAUD RATE

If :LS: has been specified as an output device, you will need to set the baud rate on switch bank 1 as shown in the BOS Users Manual. Note that CIS COBOL always transmits with no parity, 8 data bits and one stop bit (i.e. only the baud rate is read from the switches).

APPENDIX G

EXAMPLE OF USER RUN TIME SUBROUTINES

```

;*****
;*
;*
;* THIS IS AN EXAMPLE OF USER CALL CODE SUPPLIED PURELY FOR GUIDANCE OF THE
;* USER TO ENABLE THE MECHANICS OF CALL CODE INSERTION TO BE BETTER
;* UNDERSTOOD.
;* THE CODE IS DESIGNED TO BE A USEFUL EXAMPLE OF CALL, AND IF IMPLEMENTED
;* WILL ALLOW THE COBOL PROGRAMMER TO CREATE 16 BIT BINARY QUANTITIES FROM
;* UP TO 5 ASCII DIGITS, AND VICE VERSA. THE USE IS EXPLAINED IN MORE DETAIL
;* AT THE HEAD OF EACH ROUTINE.
;*
;* MICRO FOCUS LTD. HAS TAKEN EVERY PRECAUTION TO ENSURE THE ACCURACY OF
;* THESE ROUTINES, BUT CANNOT BE HELD LIABLE IN ANY WAY FOR ANY ERRORS OR
;* OMISSIONS IN THEM.
;*
;*****
;* THE MODULE MUST BE LOCATED AT THE ADDRESS SPECIFIED BY CONFIGURATOR
;* WHEN THE RTS IN WHICH THE CODE IS TO RESIDE WAS CONFIGURED. (SEE
;* OPERATING GUIDE, SECTION 5).
;*
BASE:    EQU        04404H                ;REPLACE 04404H BY THE ADDRESS
                                           ;GIVEN BY CONFIGURATOR.
;*
;*
        ORG        BASE                ;SET THE BASE ADDRESS
;*
;*
;* NOW FOLLOWS THE CALL CODE IDENTIFICATION TABLE. THIS IS A TABLE OF
;* ADDRESSES OF THE ENTRY-POINTS TO THE ROUTINES. PRECEDED BY A BINARY
;* 8 BIT ITEM SPECIFYING THE HIGHEST AVAILABLE ROUTINE NUMBER
;*
;*
CALTOP:  DB        MAXNO                ;HIGHEST AVAILABLE CALL ROUTINE.
        DW        0                    ;CALL "00" (DOES NOT EXIST)
        DW        DECBIN               ;CALL "01" - DECIMAL
ASCII TO BINARY
        DW        BINDEC               ;CALL "02" - BINARY TO
DECIMAL ASCII
MAXNO:   EQU        ($-CALTOP-3)/2      ;LET THE ASSEMBLER DO THE WORK
;*
;*
;* NB. ALTHOUGH THE USE OF CALL "00" IN THE ABOVE EXAMPLE WOULD CAUSE
;* THE RTS TO ISSUE THE FOLLOWING ERROR:-
;*
        164 - CALL CODE DOES NOT EXIST
;* THE USER IS AT LIBERTY TO PROVIDE HIS OWN CODE. BY PLUGGING IN
;* THE APPROPRIATE ROUTINE ADDRESS.
;*
;* SIMILARY, OTHER ROUTINES MAY BE ADDED BY INCREASING THE NUMBER
;* OF ADDRESSES SPECIFIED. IF THESE ARE ADDED BEFORE THE MAXNO EQUATE.
;* THEN THE BYTE AT CALTOP WILL ALWAYS BE CORRECT
;*

```

```

;*ROUTINE:          DECBIN
;*
;*CALLING SEQUENCE:
;*                CALL "01" USING PARA1 PARA2 PARA3.
;*
;*FUNCTION:        THIS ROUTINE CONVERTS A STRING OF DECIMAL (ASCII)
;*                DIGITS INTO A 16 BIT BINARY QUANTITY. IT IS VERY LOW LEVEL
;*                IN THAT IT EXPECTS A POSITIVE DECIMAL VALUE
;*
;*PARAMETERS:     PARA1 - ADDRESS OF LENGTH OF DECIMAL STRING
;*                HELD AS 1 BYTE ASCII DIGIT (NOT CHECKED)
;*                THIS ADDRESS WILL BE NO. 2 ON STACK
;*
;*                PARA2 - ADDRESS OF DECIMAL STRING
;*                THIS ADDRESS WILL BE IN B,C ON ENTRY
;*
;*                PARA3 - ADDRESS OF RESULT AREA.
;*                SPECIFIES A 2 BYTE AREA
;*                THIS ADDRESS WILL BE IN D,F ON ENTRY
;*
;*VALUES RETURNED: 16 BIT RESULT IN PARA3
;*
;*

```

```

DECBIN:
    POP      H                ;GET RETURN ADDRESS OFF STACK
    XTHL
                                ;GET ADDRESS OF PARA1
                                ;PUTTING RETURN ADDRESS BACK.
;*
    MOV      A,M              ;PUT IT IN ACCUMULATOR
    ANI      OFH              ;CONVERT TO BINARY

    PUSH     D                ;SAVE ADDRESS OF RESULT
    PUSH     B                ;MOVE STRING REF
    POP      D                ; INTO D,E
    LXI     H,0               ;HL 1 BINARY ACCUMULATOR

DEC10:
    PUSH     PSW              ;SAVE THE COUNT
    DAD     H                ;BINARY ACCUMULATOR *2
    MOV     B,H                ; AND MOVE IT INTO B,C
    MOV     C,L                ;
                                ;
    DAD     H                ;BINARY ACCUMULATOR *4
    DAD     H                ;
                                ;
    DAD     B                ; *8 + *2 1 *10
                                ; (IE. 8X + 2X 1 10X)
                                ;-----

```

```

LDAX    D                ;GET THE DECIMAL CHAR
INX     D
ANI     OFH              ;CONVERT TO BINARY CHAR
MVI     B,OH
MOV     C,A
DAD     B                ;ACC + CHAR
POP     PSW
DCR     A                ;KEEP COUNT
JNZ     DEC10

;*
;*
;*
XCHG                    ;PUT RESULT IN D,F
POP     H                ;GET ADDRESS OF RESULT AREA
MOV     M,D              ;STORE MS BYTE
INX     H
MOV     M,E              ;STORE IS BYTE
RET

;*

```

```

;*ROUTINE:      BINDEC
;*
;*CALLING SEQUENCE:
;*              CALL "02" USING PARA1 PARA2.
;*
;*FUNCTION:     TAKES THE BINARY QUANTITY ADDRESSED BY PARA1 AND CONVERTS
;*              IT INTO A 5 DIGIT DECIMAL (ASCII) NO. THE RESULT IS PLACED
;*              IN THE AREA SPECIFIED BY PARA2.
;*
;*PARAMETERS:   PARA1 1 ADDRESS OF 16 BIT (2 BYTE) QUANTITY.
;*              WILL BE IN REG B,C ON ENTRY
;*
;*              PARA2 1 ADDRESS OF 5 BYTE RESULT AREA.
;*              WILL BE IN REG D,E ON ENTRY
;*
;*VALUES RETURNED:
;*              5 DIGIT ASCII VALUE IN PARA2.
;*

```

```

BINDEC:
PUSH    B                ;GET VALUE ADDR
POP     H                ; IN H,L
MOV     B,M              ;VALUE
INX     H                ; IN
MOV     C,M              ; B,C
LXI     H,0              ;PUSH CONSTANTS
PUSH    H                ; ON TO
LXI     H,-10            ; STACK
PUSH    H                ; FOR USE
LXI     H,-100           ; DURING

;*
PUSH    H                ; BINARY TO DECIMAL COVERSION.
LXI     H,-1000
PUSH    H
LXI     H,-10000

```

```

;*
CN25:   PUSH      H                ;D,E 1 ADDRESS OF RESULT FIELD
MVI     A,30H                    ;SET TALLY TO ASCII ZERO
CN30:   POP       H                ;GET THE CONSTANT
        PUSH     H                ;RESTORE IT
        DAD     B                ;SUBTRACT FROM SOURCE OP
        JNC    CN40              ;ITS GONE NEGATIVE
        INR     A                ;INC TALLY

        PUSH     H                ;REPLACE B,C WITH
        POP     B                ; NEW RESULT
        JMP     CN30

CN40:   POP       H                ;CLEAR CONSTANT OFF STACK
        STAX    D                ;STORE TALLY IN RESULT FIELD
        INX     D                ;INC RESULT ADDR POINTER
        POP     H                ;ANY MORE CONSTANTS ,
        MOV     A,L
        ORA     H
        JZ     CN50              ;NO - FINISH OFF
        PUSH    H                ;YES - RESTORE IT
        JMP     CN25

CN50:   MOV     A,C                ;INSERT UNITS
        ADI     BOH              ;CONVERT TO ASCII
        STAX    D
        RET                       ;RETURN
;*
;*
;*

```



APPENDIX H

SUPPLIED CALL CODE ROUTINES  
EXAMPLE

```

**CIS COBOL V3.3                CALLEX.CBL                PAGE: 0001
**
000010 IDENTIFICATION DIVISION                000F
000020 PROGRAM-ID                CALL-EXAMPLE.        000F
000030*                            000F
000040* This dummy program has been produced by Micro Focus 000F
000050* as an example of the way in which the supplied CALL 000F
000060* code routines may be used.                    000F
000070*                            000F
000080 DATA DIVISION.                        000F
000090 WORKING-STORAGE SECTION.                000F
000100 01 ROUTINE-NAMES                        000F
000110     02 DECIMAL-BINARY                    PIC X(2) VALUE "01". 000F
000120     02 BINARY-DECIMAL                    PIC X(2) VALUE "02". 0011
000130*                            0013
000140 01 PARAMETER-FIELDS                      0013
000150     02 DECIMAL-NUMBER-LENGTH              PIC 9 VALUE 4.      0013
000160     02 DECIMAL-NUMBER                      PIC 9(4) VALUE 1234. 0014
000170     02 BINARY-RESULT                      PIC X(2).           0018
000180*                            001A
000190     02 BINARY-NUMBER                      PIC X(2) VALUE X"04D2". 001A
000200     02 DECIMAL-RESULT                      PIC 9(5).           001C
000210*                            0021
000220 PROCEDURE DIVISION.                      0000
000230* The following CALL will convert the 4 digit numeric field 0000
000240* DECIMAL-NUMBER to a 16 bit binary quantity in BINARY-RESULT. 0000
000250*****                                0000
000260     CALL DECIMAL-BINARY USING DECIMAL-NUMBER-LENGTH 0000
000270     DECIMAL-NUMBER BINARY-RESULT.           0000
000280*****                                000A
000290* BINARY-RESULT now contains the binary number 04D2. 000A
000300*                            000A
000310* The following CALL will convert the 16 bit binary field 000A
000320* BINARY-NUMBER to a 5 digit DECIMAL-RESULT 000A
000330*****                                000A
000340     CALL BINARY-DECIMAL USING BINARY-NUMBER DECIMAL-RESULT. 000A
000350*****                                0012
000360* DECIMAL-RESULT now contains the value 01234. 0012
**CIS COBOL V4.2 COMPILER COPYRIGHT (C) 1978 MICRO FOCUS LTD URN AA/3999/AB
**
**ERRORS=00000 DATA=00033 CODE=00043 DICT=00188:29624                END OF LIST

```