Guia do usuário

Sect

OVE

1.1 Introduction

The purpose of this COBOL-80 User's Guide is to give you practical information about getting a COBOL-80 program up and running on your computing equipment. All the steps necessary to use COBOL-80 successfully -- compiling, loading, executing, etc. -- are carefully described in the following pages.

In this guide, examples and file names are given which are based on a SOP version of COBOL-80. If you are using another operating system, the format of commands and filenames will be slightly different. See Appendix D for a description of how COBOL is used with your operating system.

1.2.1 The COBOL Compiler

The compiler consists of a main program and four overlays. These five parts correspond to the five "phases" of compilation. The main program is always memory-resident and controls the transition from each phase to the next. The overlay portion of the main program compiles the IDENTIFICATION and ENVIRONMENT DIVISIONs. Overlay 1 is brought in to compile the DATA DIVISION. The PROCEDURE DIVISION is compiled by overlay 2. These 3 parts constitute the first pass of compilation. Their function is to create an intermediate version of the program, which is stored on the current disk in a file named STEXT.INT. Overlay 3 reads the intermediate file and creates the object code. Finally, overlay 4 allocates the file control blocks and checks certain error conditions. The intermediate file is then deleted.

1.2.2 The Runtime System

The runtime library consists of a group of subroutines that interpret the object code of your program produced by the compiler. These subroutines will be included with your object program when you perform the loading step. (See section 3 of this guide). Not all programs will require all of the library routines. The loader will search the library and automatically include the portions you need and exclude the ones you don't.

1.2.3 Utility Software

The L 8Ø linking loader is used to link COBOL object programs with the runtime system. (See section 3 of this guide). The other utilities are provided for your convenience. Each of these programs is documented in the utilities Software Manual.

1.2.4 Miscellaneous Files

SQUARO.COB is a COBOL source program that computes the square root of the number you provide. It is used to verify that you have a working version of the compiler and runtime system.

SEQCVT.COM is a special utility program that converts COBOL files from LINE SEQUENTIAL format to SEQUENTIAL format. The COBOL-80 SEQUENTIAL file format was changed when version 3.0 was released. SEQUENTIAL organization files created by earlier versions are in the format that is now known as LINE SEQUENTIAL.

1.3 Program Development Steps

Preparation of a COBOL program for execution consists of three basic steps:

1. Creating the source file with a text editor
2. Compiling with the COBOL Compiler
3. Loading with the Linking Loader L 8Ø

The source program is a file which consists of lines of ASCII text terminated by carriage-return line-feed. You can create it with ED.
that uses 7-bit ASCII character codes. Line numbers may be included in columns 1-6 of each line and these may be 8-bit ASCII codes. The compiler ignores characters other than TAB and carriage return until column 7 is reached. TAB stops assumed by the compiler are at columns 7, 17, 25, 33, 41, 49, 57, 65, and 73. All characters beyond column 73 are ignored until a CR is encountered. If you use EDIT-80, you automatically begin typing in column 7 of each inserted line.

Having created the source program file, the next step is to compile it. This is done by typing a command line that will execute the COBOL compiler and provide the name of your source file. Under SOP you must be logged-in on the disk that contains the COBOL compiler, since the compiler overlays are always read from the current disk. The following example shows a command to compile the test program SQUARO assuming drive A contains a copy of that disk.

```
A> COBOL  SQUARO.REL,TTY:=SQUARO.COB
```

This command will compile SQUARO.COB, placing the relocatable object code in a file named SQUARO.REL and printing the listing on your terminal. A shorter notation of this same command line takes advantage of default file-name extensions assumed by the compiler:

```
A>COBOL SQUARO,TTY:=SQUARO
```

The shortest notation of all uses a compilation switch to force generation of an object file that defaults to the filename SQUARO.REL:

```
A>COBOL ,TTY:=SQUARO/R
```

These three example commands all produce exactly the same results. A full description of the command line syntax is given in section 2.

Once the source program is compiled, the final step before execution is to load the program with the Linking Loader L80. This step converts your relocatable object program into an absolute version and combines it with the COBOL-80 runtime system. This absolute version is built in memory, where it may then be saved on disk, executed directly, or both. The following is a command to load SQUARO and execute it without saving the absolute version.

```
A>L80 SQUARO/G
```

L80 assumes the extension .REL for the file SQUARO that is to be loaded. Once SQUARO has completed execution, you could not execute it again without performing the load command, since the absolute version was not saved. To save the absolute version in a disk file without executing it directly, type:

```
A>L80 SQUARO/N,SQUARO/E
```

Then to execute the program, simply type:

```
A>SQUARO
```

Since the absolute version is saved, it may be executed at any time without performing the load step. To combine the 2 examples so that the absolute version is saved and then executed directly, type:

```
A>L80 SQUARO/N,SQUARO/G
```

Refer to section 3 of this guide and to the utilities Software Manual for a full description of L80 commands.

Section 2

COMPILING COBOL PROGRAMS

2.1 C Line Syntax

The C reads your COBOL source program file as input and produces a listing ject version of your program. The command line invokes the COBC s it the names of the 3 files to use. The syntax of the line is to type C a space, followed by a command string, as described below. COBC. the disk and then examines the command string. If it is OK, compi ot, it types the message "?Command Error" followed by an asteri its for another command string. When compilation is complete, COBC. the operating system.

The f command string is:

=sourcefile

The se are the comma and the equal sign. No spaces are allowed. The terms are:

of the file to which the object program is to be written

the of the file to which the program listing is to be written

the of the COBOL program source file

Each me of a disk file or the name of a system device. The full descri depends on your operating system. For CP/M, a file description has th

file .extension

Here the separators are the colon and period, and the terms mean:

device
: the name of the system device, which can be a disk drive, terminal, line printer, or other device supported by the operating system. If the device is a disk, the filename must also be given. If not, the device name itself is the full file description. COBOL-80 recognizes the following symbolic device names:

  TTY: for the console terminal
  LST: for the system printer
  RDR: for the high-speed reader

filename
: the name of the file on disk. If filename is specified without a device, the current disk is assumed as the device.

.extension
: the extension of the filename given. If not specified, the following defaults are assumed:

  .COB for the source program file
  .PRN for the listing file
  .REL for the object program file

In the command string, the objectfile, listfile, or both may be omitted. If neither a listing file nor an object file is requested, COBOL-80 will check for errors and print the total on the console. If nothing is typed to the left of the equals sign, an object file is written on the same device with the same filename as the source file, but with the default extension for object files.

Examples:

Command String	Effect
,=PAYROLL	Compiles the source from PAYROLL.COB and produces only an error count, which is displayed on the console.
=PAYROLL	Compiles PAYROLL.COB and places the object into PAYROLL.REL. No listing is generated.
,TTY:=PAYROLL	Compiles the source from PAYROLL.COB and places the program listing on the terminal. No object program is generated.
PAYOBJ,LST:=PAYROLL	Compiles the source from PAYROLL.COB, places the listing on the printer, and places the object into PAYOBJ.REL.
PAYOBJ=B:PAYROLL	Compiles PAYROLL.COB from disk B and places the object into PAYOBJ.REL. No listing is generated.
PAYROLL,PAYROLL=PAYROLL	Compiles PAYROLL.COB, places the listing into PAYROLL.PRN, and places the object into PAYROLL.REL.

2.2 Compiler Switches

The command string may be modified by appending one or more switches, which affect the compilation procedure as described below. To add a switch, type a slash followed by the one-character switch name.

Switch	Action
R	Force the compiler to generate an object file. This shorthand notation causes the compiler to write the object file on the same disk and with the same filename as the source file, but with the default extension for object files.
L	Force the compiler to generate a listing file. As with /R, this notation causes the compiler to write the listing file on the same disk and with the same filename as the source file, but with the default extension for listing files.
P	Each /P allocates an extra 100 bytes of stack space for the compiler's use. Use /P if stack overflow errors occur during compilation (see section 2.3 below). Otherwise /P is not needed.

Examples of command strings using switches:

Command String	Is Equivalent To
,=PAYROLL/R	PAYROLL=PAYROLL or =PAYROLL
,=B:PAYROLL/L	,B:PAYROLL=B:PAYROLL
,=B:PAYROLL/R/L	B:PAYROLL,B:PAYROLL=B:PAYROLL
=PAYROLL/L/P	PAYROLL,PAYROLL=PAYROLL/P

2.3 Output Listings and Error Messages

The listing file output by COBOL-80 is a line-by-line account of the source file with page headings and error messages. The page heading line is printed 3 lines from the top of the page and is followed by 2 blank lines. Each source line listed is preceded by a consecutive 4-digit decimal number. This is used by the error messages at the end to refer back to lines in error, and also by the runtime system to indicate what statement has caused a runtime error after it occurs.

Two classes of diagnostic error messages may be produced during compilation.

Low level flags are displayed directly below source lines on the listing when simple syntax violations occur. Remedial action is assumed in each case, as documented below, and compilation continues. If a low-level error occurs, a high-level diagnostic will be generated at the end of the listing that refers to the line number attached to the low-level error, so the error count given at the end includes both classes of errors.

Flag	Reason for Flag	Remedial Action by Compiler
"QLIT"?	Faulty quoted literal	
	1. Zero length	Ignore and continue.
	2. Improper continuation	Assume acceptable.
	3. Premature end-of-file (before ending delimiter)	Assume program end.
LENGTH?	Quoted literal length over 120 characters, or numeric literal over 18 digits, or 'word' (identifier or name) over 30 characters.	Excessive characters are ignored.
CHRCTR?	Illegal character	Ignore and continue.
PUNCT?	Improper punctuation (e.g. comma not followed by a space).	Assume acceptable.
BADWORD	Current word is malformed such as ending in hyphen, or multiple decimal points in a numeric literal.	Ignore and continue.
SEQ #	Improper sequence number (includes case of out-of-order sequence number).	Accept and continue.
NAME?	Name does not begin with a letter (A - Z).	Accept and continue.
PIC = X	An improper Picture.	PIC X is assumed.
COL.7?	An improper character appears in source line character 'column' 7, where only * - / D are permissible.	Assume a blank in column 7.
AREA A?	Area A, columns 8-12, is not blank in a continuation line.	Ignore contents of Area A (assume blank).

High level diagnostic messages consist of two or three parts:

1. The associated source line number — four digits, followed by a colon (:).

2. An English explanation of the error detected by the compiler. If this text begins with /W/, then it is only a warning; if not, it is an error sufficiently severe to inhibit linkage and execution of an object program.

3. (Optional) The program element cited at the point of error is listed. Design of the high level diagnostic message text is such that no list of 'messages and error codes' is necessary; the messages are self-explanatory.

Regardless of whether there is a list device, or what the list device may be physically, a message displaying the total number of errors or warnings is always displayed on the console at the end of compilation. This allows you to make a simple change to a COBOL program, recompile it without a listing and still know whether the compiler encountered any questionable statements in the program.

Two error messages that occur infrequently and are also displayed on the console must be noted. One is "?Memory Full" which occurs when there is insufficient memory for all the symbols and other information the compiler obtains from your source program. It indicates that the program is too large and must be decreased in size or split into separately compiled modules. The symbol table of data-names and procedure-names is usually the largest user of space during compilation. All names require as many bytes as there are characters in the name, and there is an overhead requirement of about 10 bytes per data-name and 2 bytes per procedure-name. On the average, each line in the DATA DIVISION requires about 14 bytes of memory during compilation, and each line in the PROCEDURE DIVISION requires about 3 1/4 bytes.

The other error message, "?Compiler Error", occurs when the compiler becomes confused. It is usually caused by one of two problems: either the stack has overflowed, in which case using the /P switch will solve it; or the compiler or one of the overlay files on the disk has been damaged, in which case you should try your backup copy. If neither of these solutions works, you can sometimes determine the cause by compiling increasingly larger chunks of your program, starting with only a few lines, until the error recurs. These two error conditions cause immediate termination of compilation.

2.4 Files Used by COBOL-80

In addition to the source, listing and object files used by COBOL-80, the following files should be noted.

First, there is a file called STEXT.INT which the compiler always places on the current disk. It is used to hold intermediate symbolic text between pass one and pass two of the compiler. It is created, written, then closed, read, and then deleted before the compiler exits. Consequently, you should never run into it unless the compilation is aborted.

Another file of concern is the file to be copied due to a COPY verb in the COBOL program. (See the discussion of COPY in the COBOL-80 Reference Manual). Remember that copied files cannot have COPY statements within them and the rest of the line after a COPY statement is ignored.

Finally COBOL programs that use segmentation cause the loader to create a file for each independent segment of the program. The filenames are formed as follows. The filename itself is the PROGRAM-ID defined in the IDENTIFICATION DIVISION. The extension is .Vnn where nn is a two-digit hexadecimal number that is the segment number minus 49.

Section 3

LOADING COBOL PROGRAMS

The Linking Loader LINK-80 is used to convert the compiled relocatable object version of your program into an absolute version that is executable. It automatically combines the required portions of the COBOL runtime system with your object program. The loader is also used to link one or more subprograms together with a main program. These subprograms may be specified individually or extracted from a library, and may be written in COBOL, FORTRAN-80

3.1 LINK-80 Command Line Syntax

The complete syntax for LINK-80 commands is given in Chapter 4 of the utilities Software Manual. However, some functions described there are not useful when loading COBOL programs. This section summarizes use of the loader for COBOL programs.

You may invoke LINK-80 in one of two ways: either type L80 followed by a carriage return and enter a command string when the asterisk prompt is typed, or type L80 followed by a space, followed by the command string on the same line.

The command string is a list of filenames separated by commas. Each filename specified is brought into memory by the loader and placed at the next available memory address. Switches are used in the command string to specify functions the loader is to perform. The command string may be broken up into many small strings and entered on different lines. The loader will prompt with an asterisk and wait for more command strings until one with a /G or /E switch has been processed and the loader exits to the operating system. Filenames are specified in the same manner as for the compiler, except that the default extension is always .REL for files to be read by the loader. Such files are all expected to be in relocatable object format, so they must have been previously compiled (or assembled).

Switches most useful when loading COBOL programs are:

Switch	Effect
filename/N	Directs the loader to save the executable program on disk with name <filename> when the loading process is complete.
/E	Directs the loader to complete the loading process and exit to the operating system. The loader searches COBLIB.REL and CRTDRV.REL to resolve undefined global symbols. The final step is to save the executable program on disk, provided that the /N switch was specified.
/G	Directs the loader to complete the loading process and begin execution of the program. As with /E, the COBOL runtime library is searched, and the executable program is saved if /N was specified.

Switches used occasionally when loading COBOL programs are:

Switch	Effect
/R	Immediately resets the loader to its initial state. The effect is as if the loader was aborted and then reloaded from disk.
filename/S	Directs the loader to search <filename> to resolve undefined global symbols. This command is used to selectively load CALLed subroutines from a user-built library.
/M	Prints a map of all global symbols and their values. Undefined globals appear with an asterisk after the name.
/U	Prints a list of all undefined global symbols.

Examples:

Command String

MYPROG,SVPROG/N/E

Loads MYPROG.REL, saves the absolute version in SVPROG.COM and exits to the operating system.

MYPROG/G

Loads MYPROG.REL and begins execution without saving the absolute version.

MYPROG,SUBPR1,B:SUBPR2,MYPROG/N/E

Loads MYPROG.REL,SUBPR1.REL, and B:SUBPR2.REL. Saves the absolute version in MYPROG.COM and exits to the operating system.

MYPROG/N/S:MYLIB/S/E

Loads MYPROG.REL searches MYLIB.REL for subroutines referenced by "CALL" statements, saves the absolute version in MYPROG.COM and exits to the operating system.

3.2 Segmentation

If you have divided your program into a main module and one or more subprogram modules, you can combine them into one executable program. Before loading, compile all modules so that you have a relocatable object version of each. Then enter the loader and specify in the command string the name of each module you want to load. Modules may be specified in any order. For example, if you have a compiled main program file MAINPG.REL and 2 subprogram files SUBPR1.REL and SUBPR2.REL, you may load the executable program and save it with any of the following load command strings:

1. A>LOAD MAINPG/N,MAINPG,SUBPR1,SUBPR2/E

2. A>LOAD
*MAINPG/N,MAINPG,SUBPR1,SUBPR2
*/E

3. A>LOAD SUBPR1,SUBPR2
*MAINPG/N
*MAINPG/E

Section 4

EXECUTING COBOL PROGRAMS

You may execute a COBOL program in any of three ways. The first is to use the G switch in the loader command string as described in section 3.1. The second is simply to type the name of an executable program file as saved by using the N switch in the loader command string. Finally, you may execute a program directly from within another COBOL program by using the CALL or CHAIN statement. Refer to Chapter 5 of the COBOL-80 Reference Manual for an explanation of program CALL and CHAIN.

4.1 The Runtime System

The relocatable version of your program produced by the compiler is not 8080 or Z80 machine instructions. it is in the form of a special object language designed specifically for instructions. The COBOL-80 runtime system executes your program by each object language instruction and performing the function required. The loader all processing needed to handle CRT, printer, and disk file input and output.

The runtime system consists of a number of machine language subroutines collected into a library named COBOL and a CRT driver named CRTDRV.REL. When you load your COBOL program, COBOL is automatically searched by the loader to find and load routines that are required to perform the instructions in your source program. The number of routines needed depends on the number of COBOL language features you have used in your program and subprograms. If DISPLAY or ACCEPT statements are included in the source program, the loader automatically searches the file CRTDRV.REL to include the terminal dependent functions.

The amount of memory required by a COBOL program at runtime equals the amount required to store the data items defined in the DATA DIVISION, plus about 500 bytes per file, plus about 12 bytes per line of the PROCEDURE DIVISION, plus up to 24K bytes for the runtime system.

4.2 Printer File Handling

Printer files should be viewed simply as a stream of characters going to the printer. Records should be defined simply as the fields to appear on the printer. No extra characters are needed in the record for carriage control. Carriage return, line feed and form feed are sent to the printer as needed between lines. Note however, that blank characters (spaces) on the end of a print line are truncated to make printing faster.

No "VALUE OF" clause should be given for a PRINTER file in the FD, but "LABEL RECORD IS [illegible]" must be specified. The BLOCK clause must not be used for printer files.

4.3 Disk File Handling

Disk files must have "LABEL RECORD IS STANDARD" declared and have a "VALUE OF" clause that includes a File ID. File ID formats are described in the Utility Software Manual. Block clauses are checked for syntax but have no effect on any type file.

The format of [illegible] SEQUENTIAL organization files is that of a two-byte count of the record length followed by the actual record, for as many records as exist in the file. The LINE SEQUENTIAL [illegible] organization has the record followed by a carriage return/line feed delimiter, for [illegible] records as exist in the file. Both organizations pad any remaining space of the last physical block with control-Z characters, indicating end-of-file. To make maximum use of disk space, records are packed together with no unnecessary bytes in between.

The format of RELATIVE files is always that of fixed length records of the size of the largest record defined for the file. No delimiter is needed, and therefore none is provided. Deleted records are filled with hex value '00'. Additionally, six bytes are reserved at the beginning of the file to contain system bookkeeping information.

The format of INDEXED files is too complicated to include in this document. It is a complex mixture of keys, data, linear pointers, deletion pointers, and scramble-function pointers. It is doubtful that the COBOL programmer would require access to such information.

4.4 CRT Handling

4.4.1 Terminal Output

All output to the terminal is done by the DISPLAY statement. Characters are sent one at at time by the DISPLAY runtime module or by the CRT driver. If no cursor positioning was specified for any of the displayed items, a carriage-return and line-feed are sent following the last displayed item. Otherwise, no assumptions about carriage control are made by the DISPLAY module.

4.4.2 Keyboard Input

All input from the keyboard is done by the ACCEPT statement. One of two methods of input are used, depending on the type of ACCEPT being performed.

For a format 2 ACCEPT, a full line of input is typed, using the operating system facilities for character echo and input editing, ending with a carriage return. For this type, the character codes defined in the CRT driver have no effect.

For a format 3 or 4 ACCEPT, each character typed is read directly by the runtime ACCEPT module by using a call to the operating system. The ACCEPT module performs all necessary character echo and input editing functions. The editing control characters, function keys, and terminator keys are defined in the CRT driver.

4.5 Runtime Errors

Some programming errors cannot be detected by the compiler but will cause the program to terminate prematurely when it is being executed. Each of those errors produces a four-line synopsis, printed on the console.

```
** RUN-TIME ERR:
reason (see list below)
line number
program-id
```

The possible reasons for termination, with additional explanation, are listed below.

REDUNDANT OPEN	Attempt to open a file that is already open.
DATA UNAVAILABLE	Attempt to reference data in a record of a file that is not open or has reached the "AT END" condition.
SUBSCRIPT FAULT	A subscript has an illegal value (usually, less than 1). This applies to an index reference such as I + 2, the value of which must not be less than 1.
INPUT/OUTPUT	Unrecoverable I/O error, with no provision in the user's COBOL program for acting upon the situation by way of an AT END clause, INVALID KEY clause, FILE STATUS item, DECLARATIVE procedure, etc.
NON-NUMERIC DATA	Whenever the contents of a numeric item does not conform to the given PICTURE, this condition may arise. You should always check input data, if it is subject to error (because "input editing" has not yet been done) by use of the NUMERIC test.

PERFORM OVERLAP	An illegal sequence of PERFORMs as, for example, when paragraph A is performed, and prior to exiting from it another PERFORM A is initiated.
CALL PARAMETERS	There is a disparity between the number of parameters in a calling program and the called subprogram.
ILLEGAL READ	Attempt to READ a file that is not open in the input or I-O mode.
ILLEGAL WRITE	Attempt to WRITE to a file that is not open in the output mode for sequential access files, or in the output or I-O mode for random or dynamic access files.
ILLEGAL REWRITE	Attempt to REWRITE a record in a file not open in the I-O mode.
REWRITE; NO READ	Attempt to REWRITE a record of a sequential access file when the last operation was not a successful READ.
REDUNDANT CLOSE	Attempt to close file that is not open.
OBJ. CODE ERROR	An undefined object program instruction has been encountered. This should occur only if the absolute version of the program has been damaged in memory or on the disk file.
FEATURE UNIMPL.	An object program instruction that calls for an unimplemented feature has been encountered. This should occur only because of a damaged object program.
GO TO. (NOT SET)	Attempt to execute an uninitialized alterable paragraph containing only a null GO statement.
FILE LOCKED	Attempt to OPEN after earlier CLOSE WITH LOCK.
READ BEYOND EOF	Attempt to read (next) after already encountering end-of-file.
DELETE; NO READ	Attempt to DELETE a record of a sequential access file when the last operation was not a successful READ.
ILLEGAL [illegible]	Relative file not opened for I-O.
ILLEGAL START	File not opened for input or I-O.
NO CRT DRIVER	An ACCEPT or DISPLAY statement using cursor positioning is being executed, but no CRT driver has been selected. (See Appendix A of this guide.)
SEG nn I/O	An unrecoverable read error has occurred while trying to load a segment of a segmented program. The two digits nn are the hexadecimal notation of the segment number minus 49 and match the name of the file extension (.Vnn) on the disk.

In the case of program CHAINing, error messages may be generated by the CHAIN processing module. These errors are of the form "**CHAIN: problem" and also cause termination of the program. See Appendix B for the list of CHAIN error messages.

Appendix A

INTERPROGRAM COMMUNICATION

A.1 Subprogram Calling Mechanism

It is possible for a COBOL program to call COBOL subprograms or to call FORTRAN or assembler subroutines. However, it is not possible, currently, for a FORTRAN or assembler program to call a COBOL subroutine. Therefore, this section pertains to COBOL programs which call FORTRAN or assembler subroutines. The calling sequence described below is identical to that of POLYMAX FORTRAN-80 as it calls FORTRAN or assembler subroutines.

The COBOL runtime system transfers execution to a subroutine by means of a machine language CALL instruction. The subroutine should return via the normal assembler or FORTRAN return instruction.

To call a subprogram, use the name of the subprogram in the COBOL CALL statement. If the subprogram is an assembler or FORTRAN program, the name is defined by an ENTRY, SUBROUTINE, or FUNCTION statement. The name of a COBOL subprogram is as given in the PROGRAM-ID paragraph. Then link the subprogram to the main program using LINK-80, as described in section 3.2 of this guide.

A.2 CHAIN Parameters

The parameters passed between programs with a CHAIN USING statement are stored at the highest available memory address. The memory layout is as follows, starting at the highest available address and proceeding towards location zero. First, 32 bytes are reserved for stack space. Then the first parameter in the USING list follows, preceded by its length in bytes. The parameter length is stored in two bytes, high-order byte first. The parameter itself is stored as a string of bytes in the same order as they were stored in the DATA DIVISION, beginning at the address of the length minus the length itself. Each parameter in the USING list follows in order, each preceded by its length. The CHAINed program must expect the same number and format of parameters as were passed, as no checking can be done by the compiler or runtime system.

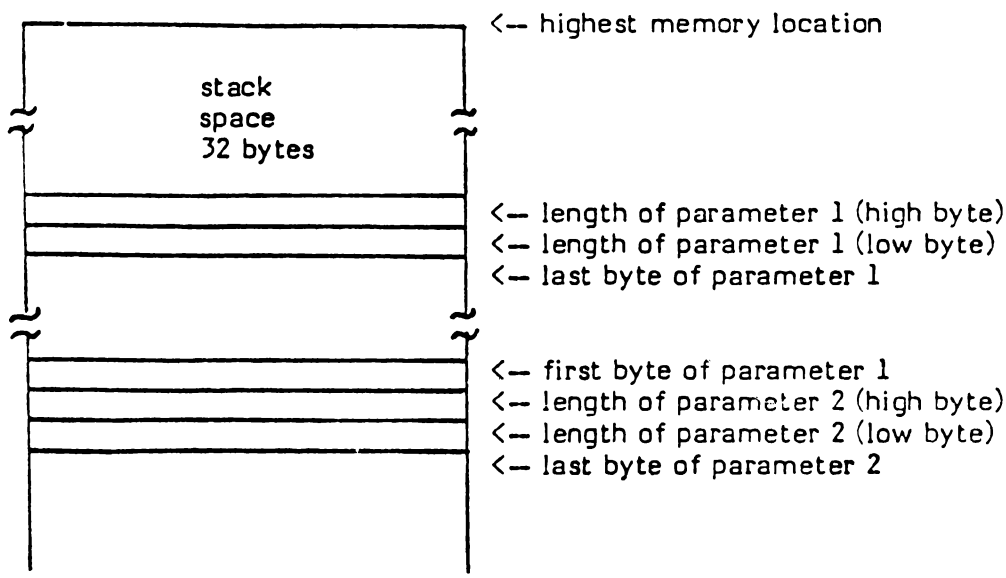

Figure B-1

A.3 CHAIN Error Messages

During CHAIN processing, the normal mechanism for reporting runtime errors may hav been overlayed by the new program. Therefore, the CHAIN processor generates its ow error messages, which are of the form "**CHAIN: problem". The following is a list d possible "problems" and their causes.

Bad file name	The syntax of the file name that is to be loaded is not valid.
File not found	The specified file was not found on the disk.
Out of Memory	There was not sufficient memory available to load the ne program. There must be enough memory for the larger of th CHAINing and CHAINed program, plus all CHAI parameters, plus 256 bytes for the program loader.

APPENDIX B

CUSTOMIZATIONS

This appendix is intended for those of you who are handy with a debugger and/or assemb language and would like to change some of the built-in parameters of COBOL-80.

B.1 Source Program Tab Stops

If tab characters (hex 09) are used in the COBOL source program, the compiler conver them into enough spaces to reach the next tab stop as defined in its internal TAB tabl As delivered, the table defines 9 stops at the following columns (counting from column 1)

7, 17, 25, 33, 41, 49, 57, 65, and 73

These may be changed by patching the table, whose address is 7 bytes from the start COBOL.COM. There is one byte in the table for each tab stop. You may supply a values you like, provided the numbers are in order and that there are still exactly 9 sto defined.

LINK-80 Linking Loader

.1 Format of LINK-80 Commands

.1.1 LINK-80 Command Strings

To run LINK-80, type L80 followed by a carriage return. LINK-80 will return the prompt "*"

indicating it is ready to accept commands. Each command to LINK-80 consists of a string of filenames and switches separated by commas:

objdev1:filename.ext/switch1,objdev2:filename.ext,...

If the input device for a file is omitted, the default is the currently logged disk. If the extension of a file is omitted, the default is .REL. After each line is typed, LINK will load or search (see /S below) the specified files. After LINK finishes this process, it will list all symbols that remained undefined followed by an asterisk.

Example:

```
*MAIN

 DATA    0100    0200

 SUBR1*      (SUBR1 is undefined)

 DATA    0100    0300

*SUBR1
*/G          (Starts Execution - see below)
```

Typically, to execute a FORTRAN and/or COBOL program and subroutines, the user types the list of filenames followed by /G (begin execution). Before execution begins, LINK-80 will always search the system library (FORLIB.REL or COBLIB.REL) to satisfy any unresolved external references. If the user wishes to first search libraries of his own, he should append the filenames that are followed by /S to the end of the loader command string.

.1.2 LINK-80 Switches

A number of switches may be given in the LINK-80 command string to specify actions affecting the loading process. Each switch must be preceded by a slash (/). These switches are:

Switch	Action
R	Reset. Put loader back in its initial state. Use /R if you loaded the wrong file by mistake and want to restart. /R takes effect as soon as it is encountered in a command string.
E or E:Name	Exit LINK-80 and return to the Operating System. The system library will be searched on the current disk to satisfy any existing undefined globals. The optional form E:Name (where Name is a global symbol previously defined in one of the modules) uses Name for the start address of the program. Use /E to load a program and exit back to the monitor.
G or G:Name	Start execution of the program as soon as the current command line has been interpreted. The system library will be searched on the current disk to satisfy any existing undefined globals if they exist. Before execution actually begins, LINK-80 prints three numbers and a BEGIN EXECUTION message. The three numbers are the start address, the address of the next available byte, and the number of 256-byte pages used. The optional form G:Name (where Name is a global symbol previously defined in one of the modules) uses Name for the start address of the program.
N	If a <filename>/N is specified, the program will be saved on disk under the selected name (with a default extension of .COM for SOP) when a /E or /G is done. A jump to the start of the program is inserted if needed so the program can run properly (at 100H for SOP).

P and D

/P and /D allow the origin(s) to be set for the next program loaded. /P and /D take effect when seen (not deferred), and they have no effect on programs already loaded. The form is /P:<address> or /D:<address>, where <address> is the desired origin in the current typeout radix. (Default radix for SOP versions is hex. /O sets radix to octal; /H to hex.) LINK-80 does a default /P:<link origin>+3 (i.e., 103H for SOP and , to leave room for the jump to the start address.

NOTE: Do not use /P or /D to load programs or data into the locations of the loader's jump to the start address 100H to 102H for SOP unless it is to load the start of the program there. If programs or data are loaded into these locations, the jump will not be generated.

If no /D is given, data areas are loaded before program areas for each module. If a /D is given, all Data and Common areas are loaded starting at the data origin and the program area at the program origin. Example:

```
*/P:200,FOO
Data    200    300
*/R
*/P:200 /D:400,FOO
Data    400    480
Program 200    280
```

U

List the origin and end of the program and data area and all undefined globals as soon as the current command line has been interpreted. The program information is only printed if a /D has been done. Otherwise, the program is stored in the data area.

M

List the origin and end of the program and data area, all defined globals and their values, and all undefined globals followed by an asterisk. The program information

is only printed if a /D has been done. Otherwise, the program is stored in the data area.

S — Search the filename immediately preceding the /S in the command string to satisfy any undefined globals.

Examples:

```
*/M
```
List all globals

```
*MYPROG,SUBROT,MYLIB/S
```
Load MYPROG.REL and SUBROT.REL and then search MYLIB.REL to satisfy any remaining undefined globals.

```
*/G
```
Begin execution of main program

.2 Sample Link

```
A>L80
*EXAMPL,EXMPL1/G
DATA    3000    30AC
[304F   30AC    49]
[BEGIN EXECUTION]

         1792          14336
        14336         -16383
       -16383             14
           14            112
          112            896
A>
```

.3 LINK-80 Error Messages

LINK-80 has the following error messages:

?No Start Address	A /G switch was issued, but no main program had been loaded.
?Loading Error	The last file given for input was not a properly formatted LINK-80 object file.
?Out of Memory	Not enough memory to load program.
?Command Error	Unrecognizable LINK-80 command.
?<file> Not Found	<file>, as given in the command string, did not exist.

%2nd COMMON Larger /XXXXXX/

The first definition of COMMON block /XXXXXX/ was not the largest definition. Re-order module loading sequence or change COMMON block definitions.

%Mult. Def. Global YYYYYY

More than one definition for the global (internal) symbol YYYYYY was encountered during the loading process.

%Overlaying [Program / Data] Area [,Start = xxxx / ,Public = <symbol name>(xxxx) / ,External = <symbol name>(xxxx

A /D or /P will cause already loaded data to be destroyed.

?Intersecting [Program / Data] Area

The program and data area intersect and an address or external chain entry is in this intersection. The final value cannot be converted to a current value since it is in the area intersection.

?Start Symbol - <name> - Undefined

After a /E: or /G: is given, the symbol specified was not defined.

Origin [Above / Below] Loader Memory, Move Anyway (Y or N)?

After a /E or /G was given, either the data or program area has an origin or top which lies outside loader memory (i.e., loader origin to top of memory). If a Y <cr> is given, LINK-80 will move the area and continue. If anything else is given, LINK-80 will exit. In either case, if a /N was given, the image will already have been saved.

?Can't Save Object File

A disk error occurred when the file was being saved.

.4 Program Break Information

LINK-80 stores the address of the first free location in a global symbol called $MEMRY if that symbol has been defined by a program loaded. $MEMRY is set to the top of the data area +1.

NOTE

If /D is given and the data origin is less than the program area, the user must be sure there is enough room to keep the program from being destroyed. This is particularly true with the disk driver for FORTRAN-80 which uses $MEMRY to allocate disk buffers and FCB's.

COBOL - 80

GUIA DE ORIENTACAO

GUIA DE ORIENTAÇÃO

Este guia de orientação é um complemento do Manual de COBOL, com o intuito de adicionar algumas informações úteis ao usuário em seu trabalho.
É um resumo de uma série de testes realizados com o compilador COBOL 80 para examinar sua performance no sistema operacional Polymax.
Os testes seguiram uma ordem de execução pré-estabelecida em função das instruções da linguagem:

- Organização de Arquivos
- Instruções de I/O
- Instruções lógicas
- Instruções aritméticas
- Considerações gerais.

Organização de Arquivos

O COBOL-80 tem as seguintes organizações de arquivos:

- SEQUENTIAL
- LINE SEQUENTIAL
- RELATIVE
- INDEXED

SEQUENTIAL

Formato

onde:
T1,T2 - são dois bytes de controle de tamanho do arquivo.
REG - é o registro que temos na FILE SECTION
Obs - não tem delimitadores OD OA

Característica

- Pode conter registros de tamanho variável (vários registros definidos na FD com tamanhos diferentes gravados intercalados).
- Se um registro é definido como FILLER não será gravado os brancos excedentes após o último caracteres não brancos.
  Ex: REG-1
  02 FILLER PIC X (80)
  Se para REG-1 for movido um campo com 40 posições diferentes de brancos, somente serão gravados 40 caracteres.
- Na regravação observar o tamanho exato do registro.
- As interrupções provocam perda da última extensão de 16k não completa.
- Perda de parte de um registro, se estiver dividido em duas extensões e a segunda for perdida.

- Um OPEN I-O em um arquivo vazio, seguido de um CLOSE, faz o retorno para o sistema sem acusar erro. Se uma operação de leitura for executada, a cláusula AT END se existir será executada.
- Quando definimos o FILE STATUS o programa espera um procedimento de recorrência antes de acusar erro.
- O maior registro pode conter 4.095 bytes.

LINE SEQUENTIAL Formato

REG - registro definido no FILE SECTION
OD,OA - delimitadores de registro.

Caracteristicas

- Tem as características idênticas a organização sequential

RELATIVE

Formato:

T1	T2	T3	T4	T5	T6	REG	REG	REG	...

onde:

T1 Indica arquivo relativo, sempre zero.
T2,T3 2 bytes indicando o tamanho do registro
T4 Não usado
T5,T6 Última chave gravada, atualizado no fechamento do arquivo
REG Registro definido na File-Section.

Caracteristicas:

Acesso Sequencial:

Após a abertura do arquivo, é reservada uma área de 2K (zeros binários), independente do tamanho do registro. Essa área é expandida de 2K em 2K conforme a necessidade.
Acesso aleatório (randômico ou dinâmico) gravação sequencial identica ao acesso sequencial.
Gravação aleatória, reserva os 2K iniciais, porém quando as chaves alocaram os registros além de 4 Kbytes iniciais do arquivo, o acesso a esses registros passa a ser realizado pelo diretório, não havendo reserva de espaço em disco para registros anteriores, ainda não gravados.
DUMP de arquivo com gravação não sequencial não lista todo o conteúdo, somente a primeira sequência.
Maior número de registros = 32.767

Interrupções:
Quando o arquivo estiver aberto para I/O e a operação for um Rewrite será colocado no primeiro byte FE, e não será possível acessar este arquivo até que o primeiro byte contenha zero novamente.

INDEXED

Formato:

O arquivo é dividido em áreas com três módulos cada.
1 módulo controle - reserva 2 setores (256 bytes)
1 módulo área de chaves - Tamanho variável
3 módulos área de registro - Tamanho variável

No primeiro módulo da primeira área estão os controles do programa.

1	indica arquivo indicado
	01 indica arquivo fechado
	FF arquivo aberto
2,3	tamanho do registro
4,5	tamánho da chave
6,7,8	não usado
9,10	no. de registro do arquivo
11,12	não usado
13,14	no. de registro do arquivo
15,256	uso do sistema

Para calcular a área de um arquivo indexado, usar o programa INDEXCOB.COM, que calcula a área ocupada em disco, a partir do tamanho do registro, tamanho da chave e número de registros.
O programa é auto explicativo.
Não existe verificação para duplicidade de chaves quando o acesso é seqüencial, a segunda ocorrência da chave é perdida.
Cláusula START aumenta a velocidade de leitura no acesso sequencial.
Interrupção provoca:

- A colocação de FF no primeiro byte do arquivo
- O arquivo só é acessado sequencialmente
  e é preciso gerar novo arquivo
- A maior chave deve ter 60 caracteres.

Comando ACCEPT sem o uso de definição da SCREEN SECTION

1. FORMATO 1 do comando ACCEPT

   1.1. ACCEPT IDENTIFICADOR-1 FROM LINE NUMBER

   IDENTIFICADOR-1 deverá ser um número inteiro sem sinal com tamanho igual a dois digitos.
   Após a execução do comando especificado, IDENTIFICADOR-1 conterá o valor correspondente à linha atual de execução do programa.

   1.2. ACCEPT IDENTIFICADOR-1 FROM ESCAPE KEY

   IDENTIFICADOR-1 conterá a chave que tenha encerrado o último comando ACCEPT (formatos 3 e 4 descritos a seguir).

   As chaves para encerramento de um comando ACCEPT estão descritas a seguir e vão retornar os seguintes valores no IDENTIFICADOR-1.

CHAVE TERMINADORA		ESCAPE KEY
BACKTAB =	CONTROL-Z encerramento apenas para formato 3	99
ESCAPE		01
FIELD-TERMINATOR	CARRIAGE RETURN, LINE FEED,	
TAB		00
FUNCTION-KEYS	Control - A	02
	Control - C	03
	Control - X	04

2. Formato 2 do comando ACCEPT

2.1. ACCEPT IDENTIFICADOR-2
Comando ACCEPT sem especificação de linha e coluna (incremento automático de linha).

3. Formato 3 do comando ACCEPT

3.1. ACCEPT (inteiro-1, inteiro-2) IDENTIFICADOR-1
Tamanho máximo do IDENTIFICADOR-1 poderá ser igual a 1.920.

Exemplo:

ACCEPT (15 , 20) IDENTIFICADOR-1

O comando ACCEPT será executado na linha 15, coluna 20

3.2. ACCEPT (LIN , COL) IDENTIFICADOR-1
LIN e COL conterão o valor da próxima linha e coluna para execução do comando ACCEPT

3.2.1. ACCEPT (LIN+1 , COL+5) IDENTIFICADOR-1
Será adicionado 1 ao valor da linha e 5 ao valor da coluna.

Obs. LIN e COL são palavras reservadas no COBOL 80

3.2.2. ACCEPT (LIN , COL) IDENTIFICADOR-1

Para alteração do valor de LIN ou COL ou para fornecer seu valor inicial, deverão ser definicas duas variáveis auxiliares comportadas e sem sinal:

Exemplo:

```
77 LINHA    PIC  99 COMP-3 VALUE 15.
77 COLUNA   PIC  99 COMP-3 VALUE 20.


PROCEDURE DIVISION
PAR.


ADD 1 TO LINHA
ADD 1 TO COLUNA

MOVE  LINHA TO LIN
MOVE COLUNA TO COL

ACCEPT (LIN , COL) IDENTIFICADOR-1
```

O comando ACCEPT será executado na linha 16 coluna 21.

3.3. ACCEPT (20 , 25) IDENTIFICADOR-1 WITH SPACE-FILL
ZERO-FILL
LEFT-JUSTIFY
RIGHT-JUSTIFY
TRAILING-SIGN
PROMPT
UPDATE
LENGTH-CHECK
AUTO-SKIP
BEEP

- ZERO-FILL preenchimento com zero à esquerda de campos numéricos.
- SPACE-FILL preenchimento com branco à esquerda de campos numéricos, se não for especificado será assumido ZERO-FILL.
- LEFT-JUSTIFY comentário.
- RIGHT-JUSTIFY ajuste à direita para campos alfanuméricos. O campo receptor deve estar descrito como "JUSTIFIELD" na SCREEN SECTION.
- TRAILING-SIGN.
  Para um item numérico com sinal descrito no formato:

```
77  NUMERO                PIC  59(4)
```

  Durante a execução do comando ACCEPT o sinal será recebido normalmente, à esquerda do número, a definição de TRAILING-SIGN, forçará a recepção do sinal à direita do campo numérico.

. PROMPT preenchimento com pontos "." o campo de entrada no vídeo.
. UPDATE o conteudo anterior do campo é exibido no vídeo para possíveis alterações.
. LENGHT-CHECK fim de campo (CARRIAGE RETURN, LINE FEED...) será ignorado a não ser que todo o campo tenha sido preenchido.
. AUTO-SKIP salto automático para o próximo campo
. BEEP sinal sonoro quando da execução de ACCEPT.

4. Formato 4 do comando ACCEPT

4.1. ACCEPT CAMPO-SCREEN ON ESCAPE Comando-Imperativo

CAMPO-SCREEN deverá ser descrito na SCREEN SECTION, se durante a entrada de dados for digitado "ESCAPE" o comando ACCEPT será encerrado sem que o campo receptor definido em CAMPO-SCREEN receba o valor digitado, ESCAPE KEY será assinalada para "01" e o Comando-Imperativo será executado.

Se, por outro lado for digitado uma "FUNCTION-KEY" (Control-A, Control-C, Control-X) ESCAPE KEY será apropriadamente assinalado (Control-A=02, Control-C=03 e Control-X=04) e o comando ACCEPT será encerrado.

ESCAPE KEY poderá ser testada através do comando já descrito anteriormente:

ACCEPT IDENTIFICADOR-1 FROM ESCAPE KEY

Se for digitado um "FIELD-TERMINATOR" (CARRIAGE-RETURN, LINE FEED, TAB) ESCAPE KEY será assinalado para "00" e o comando ACCEPT passará para o próximo campo definido em CAMPO-SCREEN.

BACKTAB (Control Z) vai ocasionar dois resultados diferentes:

No formato 3 do comando ACCEPT vai ocasionar fim de comando:

ACCEPT (LIN , COL) IDENTIFICADOR-1

No formato 4 do comando ACCEPT vai ocasionar retorno para o campo anterior dentro de DATA-SCREEN (item de grupo descrito em SCREEN SECTION)

ACCEPT DATA-SCREEN ON ESCAPE comando-imperativo

CONTROL-W retorna ao inicio do campo.

II. Descrição de um campo ou item elementar dentro de SCREEN SECTION

1. Item de grupo: Exemplo:

```
01 DATA-SCREEN   AUTO   SECURE

   02 ....
   02 ....
```

A definição de AUTO associada com um item de grupo, vai oacsionar que a cada item elementar preenchido seja executado salto automático para o próximo campo, mesmo que não tenha sido digitado um FIELD-TERMINATOR (CARRIAGE RETURN, LINE FEED, TAB).

Se for associada a definição SECURE a um item de grupo, todo item elementar a ele associado não terá seus caracteres ecoados no vídeo após a digitação, sendo trocado por "*".

2. Item elementar (MM correspondente ao número do nível do ítem elementar dentro do intervalo 01 - 49).

```
MM ITEM-SCREEN  BLANK SCREEN
          LINE NUMBER IS  PLUS  inteiro-1
          COLUMN NUMBER IS  PLUS  inteir-2
          BLANK LINE
          BELL
          HIGH LIGTH
          BLINK
          VALUE IS LITERAL-1
          BLANK WHEN ZERO
          JUST RIGHT
          AUTO
          SECURE
          DEFINIÇÃO-PICTURE
          [[FROM {LITERAL-2       }]   [TO IDENTIFICADOR-2]]
                 {IDENTIFICADOR-1 }

                 [USING IDENTIFICADOR-3]
```

2.1. BLANK SCREEN Limpar-tela

2.2. LINE NUMBER IS PLUS inteiro-1

Uma definição de um item de grupo da SCREEN SECTION poderá ter como seu primeiro item elementar uma das seguintes definições:

Exemplos:

```
01 DATA-SCREEN
   02 BLANK SCREEN
```

```
01 DATA-SCREEN
   02 LINE NUMBER 10
```

```
01 DATA-SCREEN
   02 LINE NUMBER PLUS 3
```

A primeira definição, BLANK SCREEN, vai limpar tela e posicionar o início do item de grupo (DATA-SCREEN) na primeira linha e primeira coluna, já a definição LINE NUMBER IS 10 irá posicionar o inicio do item de grupo para a décima linha.
A definição LINE NUMBER PLUS 3 irá posicionar o inicio do item de grupo na linha atual adicionada ao inteiro 3 (três).

2.3. COLUMN NUMBER IS PLUS inteiro-2

Comportamento análogo a LINE NUMBER:

```
01 DATA SCREEN
   02 COLUMN NUMBER IS 15
```

Coluna inicial igual a 15

```
01 DATA SCREEN
   02 COLUMN NUMBER IS  PLUS 4
```

Coluna atual adicionada a 4 (quatro).

As definições

LINE NUMBER IS PLUS inteiro-1

ou

COLUMN NUMBER IS PLUS inteiro-2

deverão serem usadas apenas para o primeiro item elementar de um grupo na SCREEN SECTION.

Se LINE NUMBER ou COLUMN NUMBER não forem definidos, serão considerados iguais a 1 (um).

2.4. BLANK LINE - limpar linha a partir de determinada coluna

Exemplo:

01 Limpar linha LINE 10 COLUMN 20 BLANK LINE.

o comando DISPLAY LIMPAR-LINHA irá apagar todos os caracteres da linha 10 a partir da coluna 20.

2.5. BELL - sinal sonoro

Exemplo

```
01 DATA-SCREEN
   02 BLANK SCREEN BELL
```

o comando DISPLAY DATA-SCREEN irá limpar a tela, posicionar na primeira linha, primeira coluna e emitir e sinal sonoro.

2.6. HIGHLIGHT ou BLINK - mudança da intensidade no vídeo

Exemplo

```
01 DATA-SCREEN
   02 BLANK SCREEN
   02 LINE 15 COLUMN 40 BLINK VALUE "PRIMEIRA FRASE".
```

A execução do comando DISPLAY DATA-SCREEN irá limpar a tela, posicionar na linha 15, coluna 40 e emitir com intensidade mais fraca a literal "PRIMEIRA FRASE".

2.7. VALUE IS LITERAL-1

Posicionamento de literal (entre haspas ou apóstrofes) para um item elementar

Exemplo

```
01 DATA-SCREEN
   02 BLANK SCREEN
   02 LINE 5 COLUMN 40 VALUE "ENTRADA DE DADOS".
```

2.8. BLANK WHEN ZERO

Para item numérico, zero à esquerda serão substituídos por brancos no vídeo.

2.9. JUST RIGHT

Itens alfanuméricos serão ajustadas à direita quando exibido na tela (DISPLAY)

2.10. AUTO

Salto automático de campo (se não foi definido para o item de grupo, terá validade apenas para o item elementar a que estiver associado).

2.11. SECURE

Substituição de qualquer caracter do item elementar por "*" (asterisco) quando ecoado no vídeo após a digitação.

2.12.Definição PICTURE - FROM (LITERAL-1/IDENTIFICADOR-1)

2.12.1. LITERAL-1

```
01 DATA-SCREEN
   02 BLANK SCREEN
   02 LINE 10 COLUMN 15 PIC X (10) FROM  ALL
   "=".
```

o comando DISPLAY DATA-SCREEN irá limpar a tela, e preencher a linha 10 a partir da coluna 15 com 10 (dez) caracteres iguais a "=".

2.12.2. IDENTIFICADOR-1

Exemplo

```
WORKING STORAGE SECTION
77 CA 1 PIC X(10) VALUE ALL "="

SCREEN SECTION
01 DATA-SCREEN
   02 BLSNK SCREEN
   02 LINE 10 COLUMN 15 PIC X(10) FROM CA 1.
```

A execução do comando DISPLAY DATA-SCREEN irá ocasionar uma movimentação do conteúdo do campo 1, definido na WORKING-STORAGE, para a posição 15 da linha 10 do vídeo.

2.13.Definição - PICTURE TO IDENTIFICADOR-2

Exemplo

```
WORKING-STORAGE SECTION
77 CAMPO-2   PIC X (10)

SCREEN SECTION
01 DATA-SCREEN
   02 LINE 10 PIC X (10) TO CAMPO 2
```

A execução do comando "ACCEPT DATA-SCREEN" ACCEPT DATA-SCREEN irá ocasionar uma movimentação dos 10 (dez) caracteres digitados na linha 10 coluna 1 (não houve especificação de COLUMN) para CAMPO 2 definido na WORKING-STORAGE. O conteúdo do vídeo somente será alterado na linha 10.

2.14.Definição-PICTURE USING IDENTIFICADOR 3

Exemplo

```
WORKING STORAGE SECTION
77 CAMPO,3  PIC X (10)


SCREEN SECTION
01 DATA-SCREEN
   02 COLUMN 15  PIC X (10) USING CAMPO 3
```

A execução do comando ACCEPT DATA-SCREEN vai movimentar os caracteres digitados na coluna 15, linha . 1 (não houve a especificação de LINE NUMBER) para o conteudo de CAMPO 3.

A execução do comando DISPLAY DATA-SCREEN vai movimentar o conteudo de CAMPO 3 para a coluna 15 da linha 1

3. Comando DISPLAY

O formato do comando é:

DISPLAY posição vídeo IDENTIFICADOR UPON NOME-MNEMONICO
LITERAL
ERASE

A posição no vídeo segue as mesmas normas usadas para o Comando ACCEPT.

3.1. DISPLAY (15 , 20) ERASE

A tela será limpa a partir da linha 15, coluna 20.

3.2. DISPLAY (15 , 20) DATA-SCREEN

O item de grupo DATA-SCREEN definido em SCREEN SECTION será exibido a partir da linha 15 coluna 20

3.3. DISPLAY LITERAL UPON NOME-MNEMONICO

Exemplo

```
SPECIAL-NAMES
          PRINTER IS  NOME-1


          DISPLAY "MENSAGEM" UPON NOME-1
```

A literal "MENSAGEM" será escrita na impressora.

OBSERVAÇÕES:

1. Se para um processamênto forem definidos por exemplo:

```
SPECIAL-NAMES
   DECIMAL-POINT IS COMMA


WORKING STORAGE SECTION
77 NUMERO     PIC 99V99
```

Na execução do comando

```
ACCEPT (15 , 20) NUMERO
```

deverá ser digitado "," para ajuste do ponto decimal.

Instruções Lógicas

1. Comando IF

O encadeamento do comando foi testado em dois formatos e sendo aceitos sem consição de erro até 32 IF's (não foram testados valores maiores).

1o. Formato

```
IF condição - 1
   comando composto
Else
IF condição - 2
   comando composto
Else
IF condição - 3
```

2o. Formato

```
IF condição - 1
   IF condição - 2
      IF condição - 3
         Else next sentence
      Else next sentence
   Else next sentence
Else next sentence
```

2. Comando GO TO DEPENDING ON

Para este comando foram encontradas as limitações:
O número máximo de caracteres por linha é aproximadamente 240 (= quatro linhas).
Até 20 paragrafos associados, não houve condição de erro (não foram testados valores maiores).

Instruções Aritméticas

O COBOL-80 permite cinco comandos aritméticos:
ADD, SUBTRACT, MULTIPLY, DIVIDE e COMPUTE que são complementados com as opções:

SIZE ERROR - após o alinhamento do ponto decimal e truncamento dos digitos fracionários de mais baixa ordem, o campo do resultado não tem capacidade de armazená-lo, quando ocorre o erro o resultado não é alterado se a condição está presente, caso contrário o resultado é imprevisível.

ROUNDED - após o alinhamento do ponto decimal o resultado contiver mais digitos que o campo que irá recebê-lo, haverá truncamento.
O digito manos significativo do campo resultante terá seu valor incrementado de 1, sempre que o digito mais significativo do excesso for igual ou maior do que 5.

GIVING - recebe o resultado da operação; pode ser um item de edição.

OBS.: - A divisão por zero gera a execução do comando ON SIZE ERROR.

COMANDO COMPUTE

Aceita os sinais:

+ Adição
- Subtração
* Multiplicação
/ DiVisão
** Exponenciação

Limitação: aproximadamente quatro linhas e 14 variáveis (incluindo constante e variáveis).
Todos os comandos foram testados em todas as suas combinações.

Instruções de Tabela

Para a manipulação de tabelas temos:

1. Cada tabela poderá ter a ocorrência máxima de 1.023 itens.
   01 tabela
      02 campo PIC X OCCURS 1.023 TIMES.

2. O número máximo de bytes por item será limitado pela memória disponível e pela fórmula de limitação da DATA DIVISION.

3. A limitação de 4.095 bytes por item de grupo existe apenas para definição da FILE-SECTION e SCREEN-SECTION, o que não ocorre na WORKING-STORAGE SECTION.

4. Acesso fora dos limites da tabela:
   Indice zero ou negativo, acusa o erro "SUBSCRIT FAULT"
   Para indice maior, acessa uma posição de memória a que o endereço esteja relacionado, não acusa o erro. É necessário um controle por programa do índice.

1 Limitações para a DATA DIVISION

O número de itens descritos na WORKING-STORAGE, LINKAGE e FILE SECTION segue a seguinte fórmula:

((W + 4095)/4096) + F + L + C $\leq$ 14

W tamanho da WORKING em bytes
F numero de arquivos
L numero de niveis 01 e 77 na LINKAGE
C numero de COMMUNICATIONS DESCRIPTION
não implantado ainda. Logo C = 0
A Screen-Section não é computado pela fórmula.
Se não possuir a LINKAGE (L = 0) e alterarmos disposição de fórmula teremos por aproximação:
W $\leq$ 4095 (13-F)
O número máximo ideal de arquivos é 12 por programa (embora seja permitido 14) pois:

Se F = 12
W $\leq$ 4095 (13-12) = W $\leq$ 4095
portanto temos aproximadamente 4K para WORKING

A capacidade de memória requerida pelo COBOL 80 é igual a:

W + 500F + X + RS

W número de bytes definidos para a DATA DIVISION
500F numero de arquivos (F) vezes 500 bytes
X Aproximadamente 12 bytes por linha de procedure
RS 24K bytes para RUNTIME SYSTEM

O compilador aceita o caracter TAB (09H)
Para edição de programa usar:

1 TAB + 1 espaço para margem A
2 TAB margem B

Não aceita comandos em minúscula.

Comandos para depuração de erros

Comandos de Debugging

READY TRACE

Lista na ordem de execução os paragrafos em que o programa passou.
É apresentado no vídeo o nome do paragrafo

RESET TRACE

Desativa o comando anterior.

EXHIBIT NAMED posição identificador .... UPON MNEMONICO
literal
ERASE

onde:

posição	é a posição no vídeo definida pelas normas usadas nos verbos ACCEPT e DISPLAY
IDENTIFICADOR	seu conteúdo será exibido no formato IDENTIFICADOR-1 = (conteúdo)
literal	será exibido a literal especificada
UPON MNEMONICO	usado se: ESPECIAL-NAMES. PRINTER IS MNEMONICO foi especificado.

Neste caso o resultado do comando EXHIBIT será na impressora.

Os comandos READY TRACE, RESET TRACE e EXHIBIT, quando contem a letra D na sétima coluna, só serão consideradas pelo compilador quando especificado na cláusula.

SOURCE COMPUTER WITH DEBUGGING MODE

MANUAL ROTINAS ESPECIAIS

C O B O L 8 0

SETREF
11/83

1.1 - INTRODUÇÃO

Este capítulo descreve as rotinas especiais que foram desenvolvidas para auxiliarem os programas escritos em COBOL 80. Estas rotinas são embutidas em um único módulo em disco, denominado:

LIVRO.REL (discos de 8'')
LIVRO514.REL (discos de 5'' 1/4)

Sua utilização deverá ser executada através dos utilitários L80.COM ou LD80.COM. O formato de montagem será:

L80 D:NOME1/N,D:NOME2,D:LIVRO/S/E
LD80 D:NOME1/N,D:NOME2,D:LIVRO/S/E

onde:
L80,LD80
programas de linkedição de COBOL 80 4.6 (ver guia do usuário COBOL 4.6)

D:NOME1/N
drive e o nome do programa final em código executável (.COM), que será criado.

D:NOME2
drive e nome do programa principal em código relocável (.REL), gerado pela compilação.

D:LIVRO/S ou D:LIVRO514/S
drive e módulo biblioteca; /S indica a busca das rotinas no módulo.

E/ ou G/
Para montar e terminar, ou montar e executar imediatamente.

Exemplo:

A L80 B:PROG1/N, B:PROG1, LIVRO/S/E

No exemplo acima, será criado no drive B o programa PROG1.COM, a partir do programa relocável PROG1.REL também no drive B mais as rotinas do módulo LIVRO no drive A, referenciados no programa principal.

PÁGINA INEXISTENTE

Chamada:
CALL "COMPRE" USING AREA,AREA-C,TAM.

Observação:

Esta rotina não deve ser usada com arquivos LINE-S[illegible] TA[illegible] dados comprimidos podem gerar os caracteres [illegible], [illegible], [illegible] usados como delimitadores nesta organização.

Exemplo:

```
WORKING-STORAGE
77 TAM      PIC 9(3) VALUE 30
77 AREA     PIC X(30)
77 AREA-C   PIC X(20)
PROCEDURE DIVISION
ACCEPT (10 20) AREA
CALL "COMPRE" USING AREA,AREA-C,TAM.
```

Tamanho:
386 bytes

1.3 - ROTINA DE DESCOMPRESSÃO DE DADOS

Nome:
DECOMP

Descrição:
Retorna os dados comprimidos pela rot

Parâmetros:
AREA-C
Campo que contém os dados comprimidos, [illegible]

AREA-C
Campo que contém os dados descomprimidos [illegible]
e será calculado pela fórmula:
T = 3X/2
onde X é o tamanho do campo AREA-C, e
AREA.

TAM
Campo que contém o tamanho do campo AREA-C,
como PIC 9(3).

Chamada:
CALL "DECOMP" USING AREA-C,AREA,TAM

Exemplo:

```
WORKING-STORAGE
77 TAM      PIC 9(3)
77 AREA-C   PIC X(20)
77 AREA     PIC X(30)
PROCEDURE DIVISION
CALL "DECOMP" USING AREA-C,AREA,TAM
DISPLAY (10 20) AREA
```

Tamanho:
386 bytes

"Esta folha foi deixad pro..

1.4 - ROTINA DE CÁLCULO DA DATA DE VENCIMENTO

Nome:
VENC ou DTVESD

Descrição:
Calcula a data de vencimento a partir de uma data inicial e um determinado número de dias.
A rotina DTVESD transfere a data de vencimento para segunda feira, quando o vencimento ocorrer no sábado ou domingo.
A rotina VENC não transfere a data de vencimento, caso ocorra num sábado ou domingo.

Parâmetros:
DATA-INI
Campo com a data inicial para contagem, e retorno da data de vencimento, deve ser definido como PIC 9(6).

DIAS
Campo com o número de dias que se quer calcular, deve ser definido como PIC 9(3).
Número máximo de dias: 255

CHAVE
Campo que define o início da contagem, deve ser definido como PIC 9, e inicializado com o valor 0 ou 1. Se 0, o vencimento é calculado a partir da data inicial (inclusive). Se 1, a data de vencimento é considerado fora do mês corrente. O cálculo é feito a partir do primeiro dia do mês seguinte.

Chamada:
```
CALL "VENC" USING DATA-INI,DIAS,CHAVE
CALL "DTVESD" USING DATA-INI,DIAS,CHAVE
```

Exemplo:

```
WORKING
77 DATA-INI     PIC 9(6)
77 DATA-SALVA   PIC 9(6)
77 DIAS         PIC 9(3)
77 CHAVE        PIC 9
PROCEDURE DIVISION
ACCEPT DATA-SALVA
MOVE DATA-SALVA TO DATA-INI
MOVE 90 TO DIAS
MOVE 0 TO CHAVE
CALL "VENC" USING DATA-INI,DIAS,CHAVE
DISPLAY (10 20) DATA-INI
```

Tamanho:

VENC	202 bytes
DTVESD	258 bytes

1.5 - ROTINA DE CÁLCULO DE DIAS ENTRE DUAS DATAS

Nome:
NUDIAS

Descrição:
Calcula o número de dias entre duas datas.

Parâmetros:
DATA-INI
Campo que contém a data inicial, deve ser definido como PIC 9(6).

DATA-FIM
Campo que contém a data final, deve ser definido como PIC 9(6).

DIAS
Campo que receberá o número de dias, deve ser definido como PIC 9(5).

Chamada:
CALL "NUDIAS" USING DATA-INI,DATA-FIM,DIAS

Observação:

A data inicial deve ser sempre menor que a data final.

Exemplo:

```
WORKING-STORAGE
77 DATA-INI    PIC 9(6)
77 DATA-FIM    PIC 9(6)
77 DIAS        PIC 9(5)
PROCEDURE DIVISION
ACCEPT (10 20) DATA-INI
ACCEPT (10 20) DATA-FIM
CALL "NUDIAS" DATA-INI,DATA-FIM,DIAS
DISPLAY (12 20) DIAS
```

Tamanho:
156 bytes

1.6 - ROTINA DE VERIFICAÇÃO DE DATA

Nome:
VERDAT

Descrição:
Verifica se uma data está correta, faz o cálculo para ano bissexto, aceitando nestes casos dia 29 do mês 2, caso contrário somente até 28 do mês 2.
Ano menor ou igual a zero é considerado como erro. A rotina não altera a data de entrada.

Parâmetros:
DATA-E
Campo com a data que será consistida, deve ser definido como PIC 9(6), sendo DDMMAA (dia, mês, ano).

SAIDA
Campo que conterá a indicação da verificação, deve ser definido como PIC 9. Este campo assumirá os valores 0 ou 1. Se 0, a data está correta. Se 1, a data está errada.

Chamada:
CALL "VERDAT" USING DATA-E,SAIDA

Exemplo:

```
WORKING-STORAGE
77 DATA-E    PIC 9(6)
77 SAIDA     PIC 9
     .
     .
PROCEDURE DIVISION
ACCEPT (10 20) DATA-E
CALL "VERDAT" USING DATA-E,SAIDA
IF SAIDA = 0 DISPLAY (11 20) "DATA CERTA"
  ELSE DISPLAY (11 20) "DATA ERRADA"
           .
           .
```

Tamanho:
448 bytes

"Esta folha foi deixada propositalmente em branco".

1.7 - ROTINA DE EXTENSO

Nome:
EXTEN

Descrição:
Retorna um valor numérico por extenso em cruzeiros.

Parâmetros:
LINHA
Campo que receberá o valor por extenso, deve ser definido como PIC X(180) e redefinido em 3 linhas para impressão.

VALOR
Campo com o valor numérico, deve ser definido como PIC 9(10)V99.

TAM-1
Campo com o tamanho da primeira linha, deve ser definido com PIC 9(3).

TAM-2
Campo com o tamanho da segunda linha, deve ser definido com PIC 9(3).
O tamanho da terceira linha é calculado pela rotina.

Chamada:
CALL "EXTEN" USING LINHA,VALOR,TAM-1,TAM-2

Exemplo:

```
WORKING-STORAGE
77 TAM-1       PIC 9(3) VALUE 80
77 TAM-2       PIC 9(3) VALUE 60
77 VALOR       PIC 9(10)V99
01 LINHA       PIC X(180)
01 IMPRESSAO   REDEFINES LINHA
   02 L1       PIC X(80)
   02 L2       PIC X(60)
   02 L3       PIC X(40)
    .
    .
PROCEDURE DIVISION
ACCEPT (4 10) VALOR
CALL "EXTEN" USING LINHA,VALOR,TAM-1,TAM-2
DISPLAY (5 10) VALOR
DISPLAY (6 10) L1
DISPLAY (7 10) L2
DISPLAY (8 10) L3
```

Tamanho:
2.137 bytes

1.8 - ROTINA DE CÁLCULO CHECK-DIGIT MÓDULO 11 (2 A 9)

Nome:
CHK11

Descrição:
Cálculo de check-digit, módulo 11, com pesos de 2 a 9.
O dígito é calculado da seguinte forma: supondo-se que A1 seja o primeiro byte do campo que se quer calcular o dígito, A2 o segundo, A3 ..., etc, teremos:

SOMA = A1 * 2 + A2 * 9 + A3 * 8 + A4 * 7 + A5 * 6 + A6 * 5 + A7 * 4 + A8 * 3 + A9 * 2 + A10 * 9 + A11 * 8 + A12 * 7 + A13 * 6 + A14 * 5 + A15 * 4 + A16 * 3 + A17 * 2.

Resultado = SOMA/11
Resto = Soma - (resultado * 11)
Dígito = Resto - 11

Se o valor de Resto for igual a 1 ou 0 o dígito será igual a 0.

Parâmetros:
DIGIT
Campo em que será retornado o valor do check-digit, deve definido com PIC 9.

NUMERO
Campo contendo o número que se quer calcular o dígito controle, pode ser definido com PIC 9(1) a PIC 9(17).

QUANT
Campo que contém a quantidade de dígitos do campo NUMERO, deve ser definido com PIC 9(2).

Chamada:
CALL "CHK11" USING DIGIT,NUMERO,QUANT

Tamanho:
215 bytes

"Esta folha foi deixada propositalmente em branco".

1.9 - ROTINA DE CALCULO DE CHECK-DIGIT MODULO 10 (7, 3, 1)

Nome:
CHK137

Descrição:
Cálculo do dígito de controle, com módulo 10 e pesos 7, 3 e 1. O dígito é calculado da seguinte forma: supondo que A1 seja o primeiro byte do campo que se quer calcular o dígito, o segundo, A3... etc, teremos:

SOMA = R1 + R2 + R3 + R4 + R5 + R6 + R7 ..... + R17
R1 = (A1 * 3) - 10 * (A1 * 3 / 10)
R2 = (A2 * 1) - 10 * (A2 * 1 / 10)
R3 = (A3 * 7) - 10 * (A1 * 7 / 10)
R4 = (A4 * 3) - 10 * (A4 * 3 / 10)
R5 = (A5 * 1) - 10 * (A5 * 1 / 10)
R6 = (A6 * 7) - 10 * (A6 * 7 / 10)
.
.
R17 = (A17 * 1) - 10 * (A17 * 1 / 10)

Resultado = SOMA/10
Resto = Soma - (resultado * 10)
Dígito = 10 - resto

Se resto igual a 0, dígito será igual a zero.

Parâmetros:
DIGIT
Campo que será retornado o valor do check-digit, deve ser definido com PIC 9.

NUMERO
Campo contendo o número, que se quer calcular o dígito controle, pode ser definido com PIC 9(1) a PIC 9(17).

QUANT
Campo que contém a quantidade de dígitos do campo NUMERO, ser definido com PIC 9(2).

Chamada:
CALL "CHK137" USING DIGIT,NUMERO,QUANT

Tamanho:
191 bytes

1.10 - ROTINA DE CÁLCULO DE CHECK-DIGIT MÓDULO 10 (2,1)

Nome:
CHK1021

Descrição:
Calcula o dígito de controle com módulo 10, pesos 2 e 1.
O dígito é calculado da seguinte forma: supondo-se que A1 seja o primeiro byte do campo que se quer calcular o dígito, A2 o segundo, A3 ...., etc, teremos:

SOMA = R1 + R2 + R3 + R4 + R5 + ... +R17
Se Ai 4 então Ri = (Ai * n) - 9
Se Ai 4 então Ri = Ai * n
onde: i = 1, 2, 3 ..., 17
n = 1 se i for par
n = 2 sè i for ímpar

A cada operação de soma no campo SOMA, este é comparado com 10, se maior, é subtraído de 10 até que o valor acumulado seja inferior a 10. No final das operações com os dígitos, o campo SOMA é subtraído de 10 resultando o dígito de controle.

Parâmetros:
DIGIT
Campo em que será retornado o valor do check-digit, deve ser definido como PIC 9.

NUMERO
Campo contendo o número que se quer calcular o dígito de controle, pode ser definido com PIC 9(1) a PIC 9(17).

QUANT
Campo que contém a quantidade de dígitos do campo NUMERO, deve ser definido com PIC 9(2).

Chamada:
CALL "CHK1021" USING DIGIT,NUMERO,QUANT

Tamanho:
167 bytes

"Esta folha foi deixada propositalmente em branco"

1.11 - ROTINA DE CONSISTÊNCIA CPF

Nome:
CPF

Descrição:
Realiza a consistência do dígito de controle do cadastro de Pessoa Física (CPF). O dígito é consistido da seguinte forma: supondo-se que A1 seja o primeiro byte do campo que contém CPF, A2 o segundo, A3 ... até A11 sendo os dois últimos os dígitos de controle (A10, A11), teremos:

Cálculo de A10 (primeiro dígito de controle)
SOMA = A1 *10 + A2 * 9 + A3 * 8 + A4 * 7 + A5 * 6 + A6 * 5 + A7 *
4 + A8 * 3 + A9 * 2
Resultado = SOMA/11
Resto = SOMA - (resultado * 11)
A10 = 11 - Resto

Se resto = 0 ou 1 A10 é igual a 0

Cálculo de A11 (segundo dígito de controle)
SOMA = A2 * 10 + A3 * 9 + A4 * 8 + A5 * 7 + A6 * 6 + A7 * 5 + A8
* 4 + A9 * 3 + A10 * 2
Resultado = SOMA/11
Resto = SOMA - (resultado) * 11)
A11 = 11 - Resto

Se Resto = 0 ou 1 A11 é igual a 0.

Parâmetros:
NUMERO
Campo com o número do CPF que será consistido, deve ser definido como PIC 9(11).

CHAVE
Campo que será retornado o código de validade do CPF, deve ser definido como PIC 9. Irá conter 0 se o número estiver certo, se estiver errado conterá 1.

Chamada:
CALL "CPF" USING NUMERO,CHAVE

Tamanho:
157 bytes

1.12 ROTINA DE CONSISTÊNCIA CGC

Nome:
CGC

Descrição:
Realiza a consistência dos dígitos de controle do Cadastro Geral de Contribuintes (MF) CGC. O dígito é consistido da seguinte forma: supondo-se que A1 é o primeiro byte do campo que contém o CGC, A2 o segundo, A3 ... até A14, sendo que os dois últimos são dígitos de controle (A13, A14), teremos:

1o. passo: Sobre os 12 primeiros dígitos é calculado um dígito de controle usando a rotina CHK11, CHECK-DIGIT módulo 11. O resultado é comparado com o 13o. dígito, se diferente o campo de retorno é assinalado com 1, e retorna ao programa.

2o. passo: Sobre os 13 primeiros dígitos é calculado um dígito de controle, usando a rotina CHK11. O resultado é comparado com o 14o. dígito, se diferente o campo de retorno é assinalado com 1 e retorna ao programa.

Parâmetros:
NUMERO
Campo que irá conter os dígitos do CGC, deve ser definido com PIC 9(14).

CHAVE
Campo que será retornado o código de validade do CGC, deve ser definido com PIC 9. Irá conter 0 se o número estiver correto, e se errado irá conter 1.

Tamanho:
174 bytes

"Esta folha foi deixada propositalmente em branco".

1.13 - ROTINA PARA TROCA DE DISQUETES

Nome:
TROCA

Descrição:
Permite a troca de disquetes durante a execução de um programa, atualizando o drive no Sistema Operacional.

Parâmetros:
DRIVE
Campo com o nome do drive que será atualizado, deve ser definido com PIC X. Irá conter os valores:

Drive A = A
Drive B = B
Drive C = C
Drive D = D

Chamada:
CALL "TROCA" USING DRIVE

Tamanho:
28 bytes

Observação:

Durante a execução da rotina, todos os arquivos do sistema deverão estar fechados, mesmo que não residam no drive de troca. O drive A deverá sempre conter um disquete.

"Esta folha foi deixada propositalmente em branco"

1.14 - ROTINA PARA APAGAR ARQUIVOS

Nome:
DELETA

Descrição:
Permite apagar um arquivo em disco, durante a execução de um programa.

Parâmetros:
DRIVE
Campo com o nome do drive, deve ser definido como PIC X. Irá conter os valores A, B, C, etc...

NOME
Campo que irá conter o nome do arquivo que será apagado, deverá ser definido com PIC X(11), sendo as primeiras 8 posições para o nome do arquivo e as 3 últimas o tipo.

Chamada:
CALL "DELETA USING DRIVE,NOME

Tamanho:
149 bytes

"Esta folha foi deixada propositalmente em branco"

1.15 - ROTINA PARA IMPRESSÃO EM DISCO

Nome:
IMPDSK

Descrição:
Esta rotina permite criar em disco, arquivos no formato de impressão (SPOLL), utilizando as mesmas regras e instruções do COBOL para impressão.

Parâmetros:
CHAVE
Campo com o código de execução da rotina e retorno após a execução. Deve ser definido como PIC 9. Para a chamada deve conter os valores:

0 para abertura do arquivo SPOOL (OPEN)
1 para fechamento do arquivo SPOOL (CLOSE)
2 para fechamento intermediário
3 para reinício de serviço

Para o código 2 não é necessário fornecer o nome do arquivo.
Após a chamada irá conter os valores:

0 se a operação foi bem sucedida
1 se o arquivo já existe em disco
2 se o diretório está cheio
3 reservado
4 se o disco está cheio
5 reservado
6 reservado
7 se o arquivo não existe (reinício)

NOME
Campo com o nome do arquivo. Deverá ser definido como PIC X(12), sendo:
PIC X(1) para o drive
PIC X(8) para o nome
PIC X(3) para o tipo

Chamada:
CALL "IMPDSK" USING CHAVE,NOME

Tamanho:
623 bytes

Observação:

O arquivo de impressões deve ser aberto antes da chamada "CALL" e fechado antes da chamada "CALL" para fechamento. Os códigos 2, 3 para fechamento intermediário e reinício, podem ser utilizados para proteção contra queda de força ou qualquer interrupção.

Exemplo:

```
          SELECT IMPRESSORA ASSIGN TO PRINTER
          .
          .
          .
   WORKING-STORAGE SECTION
   77 CHAVE  PIC 9
   77 NOME   PIC X(12) VALUE "AIMPRES  LIS"
ou
   01 NOME
       02 DRIVE PIC X VALUE "A"
       02 NAME  PIC X(8) VALUE "IMPRES"
       02 TIPO  PIC X(3) VALUE "LIS"
       .
       .
       .
   PROCEDURE DIVISION
   .
   .
   OPEN OUTPUT IMPRESSORA
   MOVE 0 TO CHAVE
   CALL "IMPDSK" USING CHAVE,NOME
   .
   .
   WRITE ...
   .
   .
   CLOSE IMPRESSORA
   MOVE 1 TO CHAVE
   CALL "IMPDSK" USING CHAVE,NOME
```

1.16 - ROTINA PARA INTERRUPÇÃO DE IMPRESSÃO

Nome:
BRK1

Descrição:
Permite a interrupção temporária ou definitiva de uma impressão, deve ser utilizada antes de um comando WRITE.

Parâmetros:
OPÇÃO
Campo que receberá o caracter do teclado para interrupção, deve ser definido com PIC X. Recebendo a letra:

''P'' para o processo e envia no canto inferior direito do vídeo a mensagem:
***PROCESSO PARADO

''C'' continua a impressão interrompida e emite no canto inferior esquerdo do vídeo a mensagem:
AGUARDE

''F'' permite um desvio previamente estabelecido pelo usuário, que deve testar o campo OPÇÃO.

Chamada:
CALL "BRK1" USING OPÇÃO

Tamanho:
217 bytes

Observação:

Esta rotina não executa no equipamento POLY 910 NET com impressora remota.

Exemplo:

```
PROCEDURE DIVISION
   .
   .
   .
CALL "BRK1" USING OPÇÃO
IF OPÇÃO = "F" CONDIÇÃO
WRITE ...
```

1.17 - CONVERSAO DE LETRAS EM MAIÚSCULOS/MINÚSCULOS

Nome:
CALFA

Descrição:
Converte letras maiúsculas em minúsculas e vice-versa.

Parâmetros:
TIPO
Campo que define o tipo de conversão deve ser definido como PIC 9 COMP. Se receber o valor 0, a rotina fará conversão de minúscula para maiúscula. Se receber o valor 1 fará conversão de maiúsculas para minúsculas.

TAM
Campo que contém o tamanho do campo que será convertido; deve ser definido com formato binário COM, e ter como seu valor, o tamanho do campo AREA.

AREA
Campo que receberá o texto para ser convertido, deve ser definido como PIC X.

Chamada:
CALL "CALFA" USING TIPO, TAM, AREA

O resultado estará contido no próprio campo AREA.

"Esta folha foi deixada propositalmente em branco".

1.18 - CÁLCULO DATA ANTECIPADA

Descrição:
Esta rotina realiza o cálculo de uma data antecipada em relação a uma data inicial a partir de um determinado número de dias.

A rotina ANTESD retrocede a data de antecipação para sexta-feira quando a mesma ocorrer no sábado ou domingo.

A rotina ANTECI não retrocede a data de antecipação calculada para sexta-feira se a mesma ocorrer num sábado ou domingo.

Parâmetros:
DATA-INI
Campo com a data inicial para contagem e referencia de cálculo para data de antecipação. Deverá ser definido como PIC 9(6).

DIAS
Campo com número de dias que se quer calcular. Deverá ser definido como PIC 9(3).

Observação:

O número máximo de dias não deverá ultrapassar a 255.

DATA-FIM
Campo que conterá a data de antecipação calculada no retorno da rotina para o COBOL. Deverá ser definido como PIC 9(6).

Chamada:
CALL "ANTESD" USING DATA-INI, DIAS, DATA-FIM
CALL "ANTECI" USING DATA-INI, DIAS, DATA-FIM