DEC-10-KCMC-D

PDP-10 COBOL INSTALLATION GUIDE

Date: 25-Mar-71
File: COBINS.RNO
Edition: 2

This document describes the software as of version 2.

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation.

Actual distribution of the software described in this specification will be subject to terms and conditions to be announced at some future date by Digital Equipment Corporation.

TABLE OF CONTENTS

APPENDIX

1.0 HARDWARE REQUIREMENTS FOR COBOL

The following items are mandatory for running COBOL. No optional assembly or loading parameters exist or are planned for allowing the use of COBOL on systems which do not contain this hardware:

KA10 control processor with KT10A dual relocation option

22K of user core (i.e., core in addition to monitor requirements)

800 disk blocks, available for compiler scratch files

NOTE: due to the size of the COBOL system, the software will be distributed on magnetic tape, in FAILSAFE format.

2.0 SOFTWARE REQUIREMENTS

The COBOL system requires the use of the following or later versions of DEC supported software.

MONITOR
4S.72 with reentrant capability
or 5.01 (TOPS-10) with reentrant capability

CUSPS
COMPIL version 15, to invoke COBOL and use TMPCOR
DDT version 22, previoous versions not tested with COBOL
LOADER version 52, previous version not tested with COBOL
MACRO version 44, previous version not tested with COBOL

NOTE: FUDGE2 version 11 will be needed by those assembling the compiler and doing their own modifications of the compiler if they wish to fit all of the compiler REL file on one DECtape.

NOTE: LOADER version 52 handles automatic searching of LIBOL, the run-time Library.

3.Ø ITEMS SHIPPED

The following physically distinct items constitute the release of COBOL version 2:

The PDP-1Ø COBOL Installation Guide (DEC-1Ø-KCMC-D)

2 copies of the PDP-1Ø COBOL manual (DEC-1Ø-KC1B-D)

1 level-C-format Failsafe tape of COBOL software. Specify 7 or 9 track when ordering this tape. (The COBOL software is also available on 9 DECtapes.)

4.Ø PUTTING COBOL ON YOUR SYSTEM

If you wish to put COBOL on your system and start using it without assembling and loading the whole system (a time consuming process), then merely use FAILSAFE and extract the following files:

```
*.SHR
LIBOL.REL
LIBARY.SAV
COBRG.OVR
```

If you wish to extract all the files from the tape, be forewarned that they occupy more then 7,ØØØ blocks!

5.Ø SYSTEM SOFTWARE COMPONENTS

5.1 COMPILER (COBOL)

The COBOL compiler consists of seven major phases:

COBOLA	Initialization
COBOLB	Identification and Environment Division syntax scan
COBOLC	Data Division syntax scan
COBOLD	Procedure Division syntax scan
COBOLE	code generation
COBOLF	listing
COBOLG	final assembly

In addition, there is a phase which dumps core and the contents of scratch files, whenever a catastrophic failure occurs. This phase is COBOLK.

The seven major phases and the COBOLK utility phase are grouped into six high segments:

COBOL.SHR	COBOLA, COBOLB
COBOLC.SHR	COBOLC
COBOLD.SHR	COBOLD
COBOLE.SHR	COBOLE
COBOLF.SHR	COBOLF,COBOLG
COBOLK.SHR	COBOLK

Each phase also contains an impure segment (IMPURE.MAC) and one or more of the following routines. All of these routines, except for IMPURE, are collected into the library COBOLL.REL.

ADJUST	Set up values in Data Division to match item
ALGGEN	code generator for algebraic functions
CLEANC	clean up some tables after doing DD syntax scan
CLEAND	clean up some tables after doing PD syntax scan
CLRNAM	delete entries in name table
CMNGEN	subroutines used by code generator routines
COMMON	subroutines used by all phases
DIAGS	the text of the diagnostic messages
EXPGEN	code generator for arithmetic expressions
GETASY	read AS1FIL, AS2FIL, AS3FIL
GETCPY	read CPYFIL
GETERA	read ERAFIL
GETGEN	read GENFIL
GETITM	read a source word
IFGEN	generate code for 'IF' statement
IOGEN	generate code for I/O statements
KILL	set up and call COBOLK
MOVGEN	generate code for 'MOVE' statements; also used for other code generators
MSCGEN	generate code for miscellaneous statements (e.g., examine, enter)
PATCH	a patch area (used only in debug version)
PSCAN	scan a picture string
PURAB	constants used by phases A & B
PUREC	constants used by phase C
PURED	constants used by phase D
PUREE	constants used by phase E
PURFG	constants used by phases F & G
PUTAS1	write AS1FIL
PUTAS2	write AS2FIL
PUTAS3	write AS3FIL
PUTBIN	write overlay and rel files (.OVR,.REL)

PUTCPY	write CPYFIL
PUTERA	write ERAFIL
PUTGEN	write GENFIL
PUTGET	insert entries in tables (used by syntax phases)
PUTLST	write listing file
RESVWD	constants used to replace reserved words
SQUIRL	go through syntax trees, drive syntax phases
SRTGEN	generate code for SORT statements
SRTTAB	sorts for ERAFIL and NAMTAB
STINFL	initialize a source file
TRACER	print trace of syntax phase (debug version only)
TRYNAM	look in NAMTAB for a source word
XFRGEN	generate code for control-transfer statements
XPAND	expand one of several tables
XPNPPL	check for APR traps

Major tables

There are several tables kept on the impure segment which may expand in size if necessary. The contents of these tables is:

NAMTAB	all non-literal words in the source program,, including both reserved and user words. Used to convert these words to more terse numbers.
SECTAB	the sections in the Procedure Division
DATAB	each data name
PROTAB	each procedure name
FILTAB	each file selected in the ED
MNETAB	each mnemonic name
CONTAB	each condition name (88-level)
EOPTAB	operands read from GENFIL during code generation
ALTAB	information about ALTER statements which cross segments
EXTAB	each external name, including many LIBOL routine names
VALTAB	values during DD syntax, literals during PD syntax
LITAB	values during DD syntax, literals during code generation
TAGTAB	address of special tags (i.e., %n)
TEMTAB	used during syntax scans for miscellaneous
FLOTAB	information about program flow (e.g., references to as yet undefined procedure names)

RESTAB	used during code generation to contain result operands (e.g., 'GIVING')
CPYTAB	information to help 'REPLACING' option of 'COPY' clause

Scratch files

There are several scratch files written by early phases of the compiler and read by later phases. Each scratch file has a name in the form JJJXXX.TMP, where JJJ is the user's job number, and XXX is a mnemonic to identify the file. The contents of these files is:

ERAFIL (JJJERA.TMP)	diagnostics for source errors; input to listing phase
GENFIL (JJJGEN.TMP)	output of syntax phases; input to code generator
CPYFIL (JJJCPY.TMP)	a copy of the source, with line numbers appended; input to listing phase
AS1FIL (JJJAS1.TMP)	output of syntax for ID, ED, DD; input to assembler
AS2FIL (JJJAS2.TMP)	output of syntax phases, and code generator for resident segments; input to assembler
AS3FIL (JJJAS3.TMP)	output of code generator for non-resident segments; input to assembler

5.2 Run Time System (LIBOL)

The COBOL operating system consists of subroutines used by the code generated by the COBOL compiler.

ACCEPT	reads data from TTY (ACCEPT verb)
ALPHAS	test field for ALPHABETIC
CBLIO	the I/O routines
CD6776	convert from SIXBIT to ASCII, and from ASCII to SIXBIT
COMPD	compare two 2-word computational items
COMPAR	compare two alphanumeric fields
CSORT	sort subroutines
DCV6	remove operational sign from SIXBIT character
DCV7	remove operational sign from ASCII character
DIV11	divide 1-word comp by 1-word comp
DPADD	double-precision add
DPDIV	double-precision divide
DPMUL	double-precision multiply
DPSUB	double-precision subtract

DSPFP	type a floating-point field on TTY
EDIT	move field to a field having edited picture
EXAM	EXAMINE verb
EXPON	exponentiation
FIX.	convert from floating point to double-precision computational
FLOT.1	convert from single-precision computational to floating point
FLOT.2	convert from double-precision computational to floating point
GD67	convert SIXBIT or ASCII to binary (computational)
JOBDAT	from SYS
KEY	move a non-numeric sort key to temp area
KPROG	error routine invoked when last statement in program is not a transfer of control
MAGNEG	get magnitude and negative of double-precision
MOVE	move SIXBIT or ASCII field
NEG67	determine if a SIXBIT or ASCII field is negative
NUMBRS	determine if a field is numeric
OVRLAY	overlay for non-resident segments
PD67	convert from binary (computational) to SIXBIT or ASCII
PDL	push-down list
PERF	set up PERFORM
POS67	check SIXBIT or ASCII field for POSITIVE
SETRET	grab parameters; skip returns
SIGN	move operational sign from one display field to another
SIZE1	check 1-word computational for size error
SIZ23	check 2-word computational for size error
SPACES	determine if a field is spaces
SUBSCR	evaluate subscript
UUC	UUO handler (dispatch routine)
UUODSP	dispatch table for UUO handler
ZEROES	determine if a field is zeroes

5.3 Source Library Facility

The library maintenance function, which builds and maintains a library of COBOL statements to be used by the COPY vero, is performed by LIBARY.

LIBARY inserts, deletes, and replaces groups of code in a library file, and inserts, deletes, and replaces lines of code within those groups.

Several TMP files may appear in the user's disk area after LIBARY is run. Each file has a name of the format JJJXXX.TMP, where JJJ is the user's job number, and XXX determines the contents of the file:

```
%ID - the directory of the input file
%OD - the directory of the output file
%OF - the data in the output file
%IF - the data in an intermediate input file
      (present only after a RESTART is done)
```

5.4 COBOL Report Generator (COBRG)

The COBOL Report Generator accepts as input control cards which describe report formats. This program produces as output a COBOL source program which, when compiled and loaded, produces the desired reports.

5.5 Stand-Alone Sort (SORT)

SORT accepts commands from the user's console to perform simple sorting of files. The program uses the COBOL sort subroutine (CSORT); and can be used whenever only the USING and GIVING options of the SORT verb are required.

6.0 PROGRAMMING IN COBOL

6.1 Efficient COBOL programming on the PDP-10

One basic consideration the programmer must remember is that, basically, COBOL is a language which manipulates bytes, whereas the PDP-10 is most efficient when manipulating words.

If a field described in the COBOL program occupies one or more full words, COBOL will try to generate word-move instructions (MOVE,BLT); if a field occupies only part of a word, byte instructions (LDB,DPB) are employed and consequently the program will run more slowly. In addition, when moving data from one field to another, it is best if both fields have the same usage, and both start at the same relative position within a word.

The programmer can ensure alignment of fields by using the SYNCHRONIZED clause in his data description, or by remembering that there are 6 SIXBIT (DISPLAY-6) bytes, and 5 ASCII (DISPLAY-7) bytes in each PDP-10 machine word, and setting field sizes accordingly.

A second basic consideration the programmer must remember is that COBOL is a decimal language, whereas the PDP-10 is a binary machine. The COMPUTATIONAL usage is meant to alleviate this conflict. COMPUTATIONAL items are stored in binary.

If the programmer describes a numeric field as having usage DISPLAY-6 or DISPLAY-7, COBOL will generate code to convert the data to binary before doing any arithmetic operation. This will not only result in a larger program, it will also be much slower.

For example, take the following items:

```
     77    CA    PIC  599; COMP
     77    CB    PIC  599; COMP.
     77    DA    PIC  599; DISPLAY-6
     77    DB    PIC  599; DISPLAY-7
```

the statement ADD CA TO CB would result in

```
     MOVE 1,CA
     ADDM 1,CB
```

whereas the statement ADD DA TO DB would result in

```
     GD6. 1,[POINT 6,CA]
     GD6. 3,[POINT 6,DA]
     ADD  1,3
     PD6. 1,[POINT 6,CA]
```

(GD6. and PD6. are UUO's which call subroutines to convert from SIXBIT to binary and back). Execution time for the second example is 100-500 times slower than that for the first example.

6.2 Reporting Bugs, Suggestions, Manual Errors

If any bugs, other than those noted in the documentation memo (DOCCOB) included with the sources of the compiler, are found by the user, the following steps should be taken.

1. Ensure that the problem was not caused by a source error. For example, one error in the source may produce more than one diagnostic.

2. Fill out an STR form and mail to DEC Maynard via your DEC Software Specialist. The form should include all pertinent information, e.g., the descriptive clauses of data-names involved, the exact statement in error, etc. Also include references to the COBOL manual that support your interpretation of the correct behavior.

6.3 Linking COBOL Programs to Programs Written in Other Languages

COBOL programmers interested in calling FORTRAN or MACRO subroutines are referred to Chapter Six and Appendix C of the COBOL manual for information on the ENTER verb and the calling sequences generated by the COBOL compiler.

It is not possible to create COBOL subroutines. A COBOL program will always be a main program.

7.0 I/O CONSIDERATIONS

7.1 File Formats

Definition of terms

Logical Record - the smallest unit of data that can be processed by the operating system. In COBOL, this is also called simply RECORD.

Physical Record - the smallest unit of data that can be processed by the hardware (e.g., 128 words for disk, 80 columns for card-reader, the data between record gaps for magnetic tape).

Buffer - an area of core memory into which the monitor reads, or from which the monitor writes, a physical record.

Blocking factor - that number specified in the "Block Contains" clause of the File-descriptor for the file; if there is no "Block Contains" clause, the blocking factor is said to be zero.

Logical Block - those buffers required to contain a number of contiguous records, that number being the blocking factor. A logical block may extend over many buffers, but always uses an integral number of buffers; any unused portion of the last buffer is wasted. If the smallest record of a file is much smaller than the largest record, there could be several wasted buffers, since the number of buffers required is always determined by the size of the largest record multiplied by the number of logical records contained in the logical block.

File - an ordered collection of contiguous logical records; the largest unit of data that can be processed by the operating system.

A file is considered "blocked" if the blocking factor is non-zero; it is considered "unblocked" if the blocking factor is zero.

Files are blocked for two reasons:

1. The output device is an MTA. A non-standard buffer size is used to reduce the number of interrecord gaps. The non-standard buffer size is set to contain one logical block.

2. At some time, the file is to be accessed randomly or the file is to be open for input/output processing. The blocking factor enables the operating system to precisely and efficiently locate a given record.

Data Structure

A record may be either SIXBIT (DISPLAY-6) or ASCII (DISPLAY-7) and either fixed or variable length.

SIXBIT.
A SIXBIT record is a set of contiguous words. The first word has, in the right half, the number of characters in the record. The last word may be padded to ensure that the record boundary coincides with a word boundary. The amount of buffer space required is the number of characters in the record plus six characters for the character count in the first word plus the number of padding characters.

ASCII.
An ASCII record is a set of contiguous characters terminated with a "carriage-return". If the record was generated with a COBOL WRITE without the advancing clause, a "line-feed" is also appended. If the advancing clause was used a string of 0 to 63 printer control characters either precedes or follows the record. Word boundaries have no

significance, the last character of a record is immediately followed by the first character of the next record. The amount of buffer space required is:

1. Advancing clause was used, number of characters in the record plus the number of printer control characters plus one (CR), or

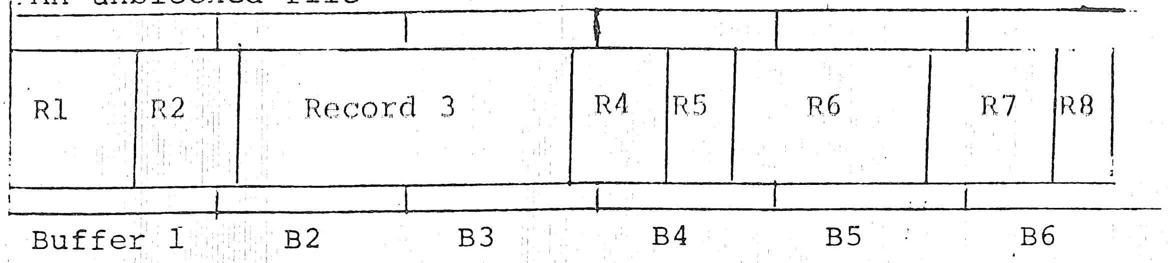
2. Advancing clause was not used, number of characters in the record plus two (CR-LF).

When reading a record, the operating system recognizes the following as "end-of-line" (EOL) characters:

ASCII code	12 - 15	LF, VT, FF, AND CR
	2Ø - 24	Printer control chars
	32	TTY EOF character CONTROL-Z
	33, 175, 176	The three flavors of ALTMODE

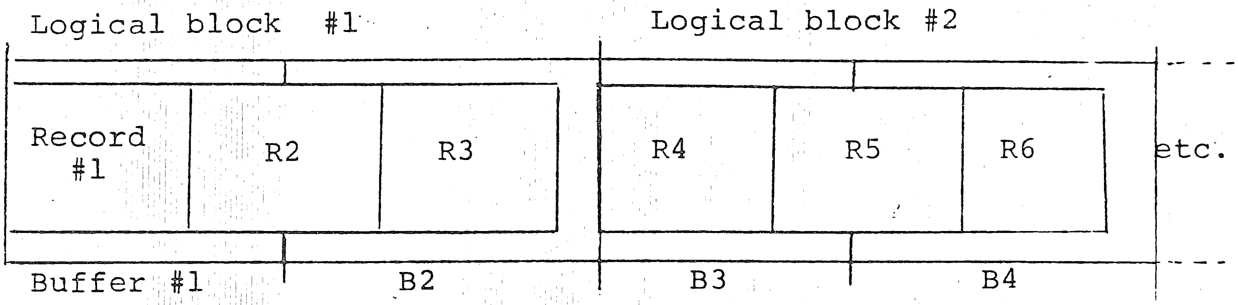
Leading EOL characters are ignored. A record terminates with the first EOL character or a satisfied character count. If the character count was satisfied before an EOL character was encountered, the remaining characters are discarded until an EOL character is encountered. If the EOL character comes before the character count is satisfied, ASCII spaces are passed until it is satisfied. ASCII null characters are always ignored.

An unblocked file

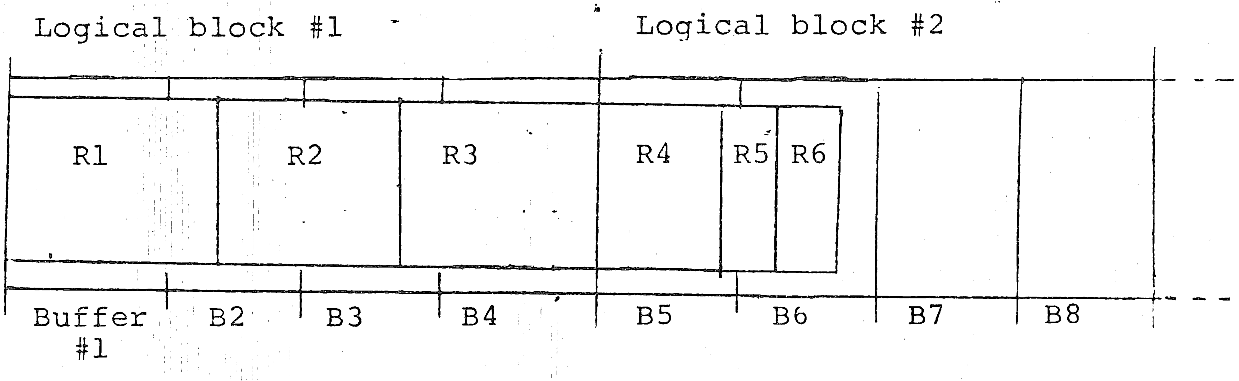


The record length is variable; the blocking factor is Ø

Blocked files

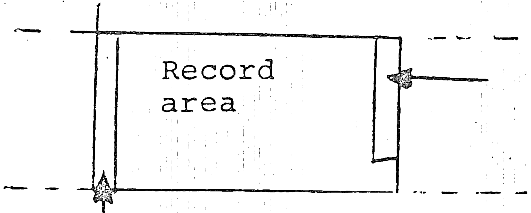


The record length is fixed; the blocking factor is 3. Note that the last portion of the last buffer of each logical block is wasted.



The record length is variable, the blocking factor is 3. The first 3 records are the maximum length; the next 3 records are much shorter causing over 2 buffers to be wasted.

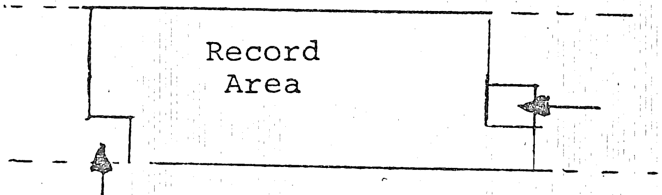
SIXBIT record



Null characters are appended to fill out the last word.

The right half of the first word contains the number of characters in the record area .

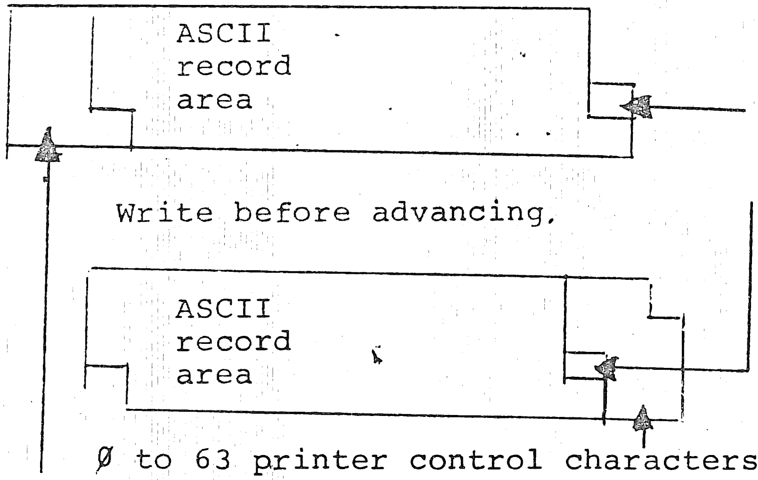
ASCII record



"CR-LF" is appended the end of the record.

ASCII records may begin and end in any character position.

Write after advancing



"CR" always immediately follows the ASCII record area

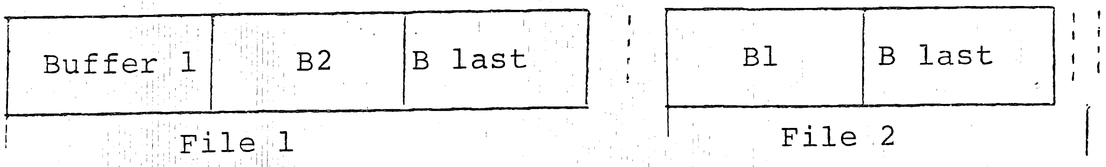
Write before advancing.

Ø to 63 printer control characters may be appended before or after the record area.

Labels

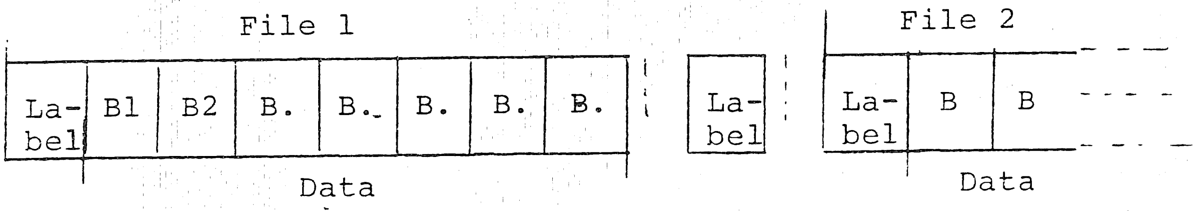
Only two devices may have labels written out with the data file, a card file and a mag-tape file. Directory devices use the directory for the label. A card file has only a "beginning-file-label" and it is the first card of the file. A mag-tape file may have 2 or more labels. If the labeled file is a multi-reel-file, it has 2 labels for each reel. A labeled file contained entirely on one reel has two labels. See the COBOL manual (DEC-10-KC1B-D), Table 8-3 for the standard label format.

Unlabeled MTA files



Files are separated by an end-of-file mark. Two end-of-file marks denote logical end of tape.

Labeled MTA files



The beginning file label occupies the first buffer of the file. The data follows immediately and is terminated with an EOF mark. The ending file label occupies the last buffer and is followed with another EOF mark.

7.2 Use of SIXBIT I/O

DISPLAY-6 (SIXBIT) files should be used only for applications where speed and efficient use of file storage are important considerations and where file compatibility with DEC software is not a concern. SIXBIT I/O is handled by COBOL only. SIXBIT I/O will not be handled by any presently implemented or planned system utilities, editors, or spoolers.

7.3 Data File Compatibility

COBOL programs can read and write files written in either ASCII or SIXBIT mode, blocked or unblocked. However, all of the other CUSPs are restricted to ASCII files; some are restricted to reading only line-blocked files (files in which no line (record) may be split between two buffers).

Sequence numbers, generated by some CUSPS (e.g., LINED) are treated as data by COBOL programs; COBOL programs will never generate sequence numbers acceptable to LINED (bit 35 of the word containing the sequence number is 1 if LINED is to accept it). However, a file generated by a COBOL program may be passed through PIP in order to line-block and sequence the file.

Following is a chart giving the characteristics of the major CUSPs.

	Sequence Numbers	Maximum Characters Per Line	Line Terminator	Line-Blocked	Special Character Processing	COBOL Compatible
FORTRAN						
Output	NO	132	CR-LF	YES		YES
Input	NO	The capacity of the storage device or core.	CR-LF	YES, req'd	ASCII Ø-1Ø, 16-24 are ignored	YES, if line-blocked
BASIC						
Output	YES Optional	132	CR-LF	NO	Nulls are ignored	YES
Input	Optional	132	CR-LF	NO		YES Versions 16 and later of BASIC allow the user to read and write files without sequence numbers.
TECO						
Output	NO	The capacity of the storage device	CR-LF	NO		YES - if the line length is not greater than 4Ø95.
Input	YES		CR-LF	NO	Nulls are ignored.	YES
LINED						
Output	YES	635	CR-LF-FF	YES		YES- sequence numbers treated as data
Input	YES, Req'd	635	CR-LF-FF	YES, Req'd	Nulls are ignored	YES - If sequence numbers are added the file is line blocked and the line length < 636.
COBOL						
Output	NO	4Ø95	ASCII:20-24 CR-LF	NO		
Input	NO	4Ø95	ASCII:12-15, 20-24, 32	YES	Nulls are ignored	

8.Ø INFORMATION FOR THE SYSTEMS PROGRAMMER

8.1 Internal Documentation

There are several memoranda describing in more detail the workings of the COBOL compiler. In particular, the following memoranda exist:

Ø1Ø	File structure for the PDP-1Ø COBOL compiler
Ø11	Table structure for the PDP-1Ø COBOL compiler
Ø17	Subroutine Calling Sequences for LIBOL

These memoranda are included as an appendix to this Guide.

8.2 Using DDT with the Compiler

In order to successfully utilize DDT to work on the compiler, one must assemble the compiler with DEBUG=1 (See COBOL.CPR memo supplied with the compiler).

Each segment must be saved so as not to be sharable (use SAVE instead of SSAVE).

Since the compiler has five high segments, breakpoints placed in one high segment will not, of course, be in the other segments. There is a global, DDTSTP in the low segment, at which control is passed to a new segment after it is loaded. A breakpoint placed at this location will allow the systems programmer to place breakpoints in the segment just loaded. Care should be taken never to have breakpoints in more than one segment, this will only confuse DDT.

The Linking Loader places symbols into the low segment, thus each low segment has only the symbols for the corresponding high segment. However, COBOL runs with only one low segment (COBOL.LOW). Therefore, if the programmer wants to check out COBOLD.HGH, for example, he must first rename COBOLD.LOW to be COBOL.LOW.

Important note: all phases must be loaded with the same LOWSEG.REL (see COBOL.OPR). This is done whether or not DDT is being used. All low segments must be identical except for the symbol table used by DDT.

8,3 Using DDT with the Object Program

COBOL will not generate local symbols, only global symbols will appear in the symbol table. This is true because COBOL words (30 characters with embedded hyphens) are not agreeable for DDT.

Global symbols include all symbols on the two pages of assembly listing starting with START.; thus the user can, with DDT, define a new symbol corresponding to location 0 of the generated program:

```
START.=y<X:
```

where y is the relative location of START.

An easier approach might be the following: load the generated program by typing to the Linking Loader

```
/DPROG,SYS:LIBOL/L$
```

Now the global DDTEND is equivalent to location 0 in PROG, and one can define

```
DDTEND<X:
```

FILE STRUCTURE FOR PDP-10 COBOL COMPILER

THE INFORMATION IN THIS MEMORANDUM
IS SUBJECT TO CHANGE WITHOUT NOTICE AND
SHOULD NOT BE CONSTRUED AS A COMMITMENT
BY DIGITAL EQUIPMENT CORPORATION

1. GENERAL INFORMATION ON FILE STRUCTURE.

THE COBOL COMPILER USES FOUR FILES SPECIFIED BY THE USER. THESE FILES MAY BE ON ANY APPROPRIATE DEVICE.

1) SRC - THE SOURCE PROGRAM

2) LST - THE LISTING

3) BIN - THE RELOCATABLE BINARY PRODUCED BY THE COMPILER

4) LIB - THE LIBRARY TO USE WITH THE COPY VERB.

THE COMPILER USES SEVERAL FILES FOR TEMPORARY DATA STORAGE. ALL OF THESE FILES ARE ON THE DISK, AND WILL BE DELETED AT THE COMPLETION OF COMPILATION. ALL FILES HAVE THE USERS JOB NUMBER AS THE FIRST THREE CHARACTERS OF THE FILE-NAME, AND AN EXTENSION "TMP". THE THREE CHARACTERS APPENDED TO THE JOB NUMBER TO FORM THE FILE-NAME ARE GIVEN BELOW WITH EACH FILE:

1) ERA - DIAGNOSTICS TO BE LISTED WITH THE SOURCE

2) GEN - OUTPUT OF THE SYNTAX PARSING PHASES

3) CPY - A COPY OF THE SOURCE FILE WITH LINE NUMBERS ASSIGNED.

4) AS1 - INTERMEDIATE LANGUAGE CONTAINING ASSEMBLY INFORMATION FOR THE IMPURE AREA OF THE OBJECT PROGRAM

5) AS2 - INTERMEDIATE LANGUAGE CONTAINING ASSEMBLY INFORMATION FOR THE PURE AREA OF THE OBJECT PROGRAM

6) AS3 - INTERMEDIATE LENGUAGE CONTAINING ASSEMBLY INFORMATION FOR THE NON-RESIDENT SEGMENTS OF THE OBJECT PROGRAM.

REFERENCES ARE MADE IN THE FOLLOWING TEXT TO VARIOUS TABLES USED BY THE COMPILER. THESE TABLES ARE DESCRIBED IN MEMORANDUM NUMBER 1ØØ-35Ø-Ø11.

2. DESCRIPTION OF THE CONTENTS OF THE FILES

2.1 SRC - THE SOURCE FILE CONTAINS THE PROGRAM TO BE COMPILED, IN ASCII.

2.1.1 THE SOURCE FILE IS SCANNED IN THE SUBROUTINE GETITM. THIS ROUTINE RETURNS INFORMATION IN ACCUMULATORS:

W1:	BIT Ø	THE ITEM IS NOT IN NAMTAB
	BIT 1	DATUM IS A LITERAL
	BIT 2	DATUM IS A RESERVED WORD
	BIT 3	LITERAL HAS A LEADING SIGN
	BIT 4	LITERAL HAS A DECIMAL POINT
	BIT 5	LITERAL IS NUMERIC
	BIT 6	"ALL" SEEN
	BIT 7	NOT USED (FOR FIG. CONST. IN SYNTAX ROUTINES)
	BIT 8	LITERAL CONTAINS NON-SIXBIT CHARACTERS
	BITS 9-17	IF RESERVED WORD, THIS IS IT'S VALUE (SEE ROUTINE RESVWD FOR VALUES)
	BITS 18-35	TABLE-LINK TO TABLE ENTRY FOR THIS ITEM (FIRST ENTRY IF MORE THAN ONE)

W2:	BIT Ø	NOT USED
	BITS 1-15	LINK TO NAMTAB ENTRY
	BITS 16-28	LINE NUMBER OF INPUT DATUM
	BITS 29-35	CHARACTER POSITION OF INPUT DATUM

CT: A CODE:
IF RESERVED WORD, THIS IS IT'S VALUE (SAME AS BITS 9-17 OF W1)
IF NOT A RESERVED WORD:

1ØØØ	USER NAME, AS YET UNDEFINED
1ØØ1	FILE NAME
1ØØ2	MNEMONIC DEVICE NAME
1ØØ3	MNEMONIC, NOT A DEVICE NAME
1ØØ4	CONDITION-NAME
1ØØ5	EXTERNAL NAME
1ØØ6	PROCEDURE NAME
1Ø1Ø	FIGURATIVE CONSTANT
1Ø11	INTEGER
1Ø12	NON-INTEGRAL NUMERIC LITERAL
1Ø13	NON-NUMERIC LITERAL
1Ø14	INDEX NAME
1Ø15	RECORD NAME
1Ø16	GROUP DATA NAME
1Ø17	OTHER DATA NAME

IF DATUM WAS IN A-MARGIN (STARTED IN POSITIONS 8-11), BIT 26 IS SET IN CT.

2.2 LST - THE LISTING IS DIVIDED INTO 3 MAJOR SECTIONS:

2.2.1 A COPY OF THE SOURCE PROGRAM, WITH IMBEDDED DIAGNOSTIC MESSAGES.

2.2.2 MAPS OF THE DATA AND PROCEDURE DIVISIONS, IF REQUESTED BY THE USER WITH THE /M SWITCH.

2.2.3 A LISTING OF THE GENERATED OBJECT CODE, IF THE USER REQUESTS IT BY USING THE /A SWITCH.

2.3 BIN - THE RELOCATABLE BINARY IS IN A FORMAT COMPATIBLE WITH THE PDP-1Ø LINKING LOADER. THE DESCRIPTION OF ITS CONTENTS CAN BE FOUND IN THE MACRO-1Ø MANUAL.

2.4 LIB - THE SOURCE LIBRARY FORMAT CAN BE FOUND IN THE MEMORANDUM DESCRIBING THE COBOL LIBRARY MAINTENANCE PACKAGE (SEE MEMO # 1ØØ-35Ø-ØØ1).

.5 ERA - THE DIAGNOSTIC FILE CONTAINS A ONE-WORD ENTRY FOR EACH ERROR DETECTED BY THE COMPILER. THE FILE IS WRITTEN BY PHASES B, C, D AND E, AND READ BY PHASE F.

BIT Ø — ALWAYS ZERO TO AID SORTING OF THE FILE.

BIT 1 — A ZERO IF THIS DIAGNOSTIC IS TO BE IMBEDDED IN THE SOURCE LISTING, A ONE IF THE DIAGNOSTIC IS TO BE LISTED SEPERATELY.

BITS 2-14 — THE LINE NUMBER OF THE WORD FOUND TO BE IN ERROR. THIS IS USED TO DETERMINE WHERE IN THE LISTING THE DIAGNOSTIC MESSAGE IS TO APPEAR.

BITS 15-21 — THE CHARACTER POSITION UNDER WHICH TO PUT THE "↑" CHARACTER IN THE LISTING.

BITS 22-25 — THE COMPILER PHASE WHICH GENERATED THE DIAGNOSTIC. (1=PHASE A,...,5=PHASE E).

BIT 26 — A ZERO IF THIS IS A WARNING, A ONE IF THIS IS A FATAL ERROR.

BITS 27-35 — A NUMBER IDENTIFYING THE ERROR MESSAGE. THE MESSAGES ARE GIVEN IN MEMO #1ØØ-35Ø-Ø16.

2.6 GEN - THE GENERATOR INPUT CONSISTS OF OPERATORS AND OPERANDS PUT OUT BY THE SYNTAX PARSING PHASES B,C AND D. EACH DATUM IS TWO WORDS.

2.6.1 OPERATORS

WORD 1:

BIT Ø — A Ø TO IDENTIFY THIS AS AN OPERATOR.

BITS 1-8 — A CODE IDENTIFYING THE OPERATOR (SEE 2.6.4)

BITS 9-15 — FLAGS REQUIRED FOR CODE GENERATION (SEE 2.6.5)

BITS 16-28 — A SOURCE LINE NUMBER USED WHEN ANY ERRORS ARE FOUND (SEE 2.5)

BITS 29-35 — A CHARACTER POSITION USED WHEN ANY ERRORS ARE FOUND (SEE 2.5)

WORD 2:

BOTS Ø-27 — NOT USED

BITS 28-35 — A CODE IDENTIFYING THE OPERATOR (COPIED FROM BITS 1-8 OF WORD 1).

'GEN CONT'D)

2.6.2 OPERANDS, OTHER THAN LITERALS.

WORD 1:

BIT Ø	A 1 TO IDENTIFY THIS AS AN OPERAND.
BIT 1	A Ø TO IDENTIFY THIS AS OTHER THAN A LITERAL.
BITS 2-4	USAGE (SEE CODES IN MEMO 1ØØ-35Ø-Ø11, PARA.4.3, WORD 5, BITS 15-17)
BIT 5	SYNCHRONIZED LEFT
BIT 6	SYNCHRONIZED RIGHT
BIT 7	NUMERIC (1) OR NON-NUMERIC (Ø)
BIT 8	JUSTIFIED LEFT (Ø) OR RIGHT (1)
BITS 9-15	IF THIS OPERAND IS A TEMPORARY PREVIOUSLY REFERENCED IN A "STASH" (SEE 2.6.4), THIS FIELD CONTAINS A NUMBER IDENTIFYING THAT TEMPORARY.
BITS 16-35	LINE NUMBER AND CHARACTER POSITION (SEE 2.6.1)

WORD 2:

BIT Ø	IGNORE TRUNCATION ERRORS
BIT 1	"ROUNDED" CLAUSE PRESENT (VALID ONLY WITH "RESULT" OPERATOR).
BIT 2	OPERAND REFERENCES FLCTAB (SEE MEMO 1ØØ-35Ø-Ø11)
BITS 3-4	NOT USED
BIT 5	OPERAND IS TEMP OR AC'S (SET BY PHASE E)
BITS 6-7	ALWAYS Ø
BITS 8-11	NOT USED
BITS 12-17	THE NUMBER OF FOLLOWING ENTRIES WHICH ARE TO BE USED AS SUBCRIPTS TO THIS OPERAND.
BITS 18-2Ø	THE TYPE OF OPERAND: Ø - FILE NAME 1 - DATA NAME 2 - CONDITION NAME 3 - LITERAL 4 - PROCEDURE NAME 5 - EXTERNAL NAME 6 - VALUE 7 - MNEMONIC NAME
BITS 21-35	THE ADDRESS OF THE ENTRY FOR THIS OPERAND RELATIVE TO THE BEGINNING OF THE TABLE WHOSE TYPE IS IN BITS 18-2Ø.

(GEN CONT'D)

2.6.3 LITERAL OPERANDS

WORD 1:

BIT Ø	A 1 TO IDENTIFY THIS AS AN OPERAND.
BIT 1	A 1 TO IDENTIFY THIS AS A LITERAL OR FIGURATIVE CONSTANT
BIT 2	NUMERIC (1) OR NON-NUMERIC (Ø)
BIT 3	A 1 IF THIS IS A FIGURATIVE CONSTANT
BIT 4	A 1 IF LITERAL CONTAINS NON-SIXBIT CHARACTERS
BITS 5-7	ALWAYS Ø
BIT 8	"TODAY"
BIT 9	"TALLY"
BIT 1Ø	"SPACE", "SPACES"
BIT 11	"ZERO","ZEROS","ZEROES"
BIT 12	"QUOTE","QUOTES"
BIT 13	"HIGH-VALUE","HIGH-VALUES"
BIT 14	"LOW-VALUE","LOW-VALUES"
BIT 15	"ALL"
BITS 16-35	LINE NUMBER AND CHARACTER POSITION (SEE 2.6.1).

WORD 2:
THIS WORD CONTAINS NO DATA IF THE OPERAND IS A FIGURATIVE CONSTANT; OTHERWISE

BITS Ø-17	SIZE OF LITERAL, IN WORDS
BITS 18-35	TABLE-LINK TO VALTAB ENTRY

(GEN CONT'D)

2.6.4 VALUES FOR THE OPERATORS

CODE	NAME	OPERANDS	USE
ØØ1	MOVE	A,B,...	MOVE A TO B,...
ØØ2	ADD	A,B,...	ADD A,B... (GIVING...)
ØØ3	ADDTO	A,...	ADD A,... (TO...)
ØØ4	SUB	B,A,...	SUBTRACT A,... FROM B (GIVING...))
ØØ5	SUBFRM	A,...	SUBTRACT A,... (FROM...)
ØØ6	MUL	A,B	MULTIPLY A BY B (GIVING...)
ØØ7	MULBY	A	MULTIPLY A (BY...)
Ø1Ø	DIV	A,B	DIVIDE A BY B (GIVING...)
Ø11	RESULT	A,...	GIVING (FROM,TO,BY) A,...
Ø12	REMAIN	A	REMAINDER A
Ø13	DIVBY	A,B	DIVIDE A BY B (NO GIVING)
Ø2Ø	IF	A,B	IF A (=, <, >) B
Ø21	IFC	A	IF CONDITION-NAME-A
Ø22	IFT	A	IF NUMERIC, POSITIVE,...
Ø23	SPIF	-	AT END, INV.KEY, SIZE ERROR
Ø24	ELSE	-	BEGINNING OF ELSE PATH FOR CONDITIONAL
Ø26	ENDIF	-	END OF CONDITIONAL
Ø3Ø	GO	A	GO TO A
Ø31	GODEP	B,A,...	GO TO A,... DEPENDING ON B
Ø32	PERF	A,[B]	PERFORM A [THRU B]
Ø33	PRFTYM	C,A,[B]	PERFORM A [THRU B] C TIMES
Ø34	ALTER	A,B	ALTER A TO PROCEED TO B
Ø4Ø	STOP	[A]	STOP RUN, STOP A
Ø42	EXAM	A,B,[C]	EXAMINE A TALLYING B [REPLACING C]
Ø43	SETTO	B,A,...	SET A,... TO B
Ø44	SETDN	B,A,...	SET A,... DOWN BY B
Ø45	SETUP	B,A,...	SET A,... UP BY B
Ø46	USING	A,...	(ENTER B) USING A,...
Ø47	ENTER	A	ENTER

Ø5Ø	COMPUT	B	COMPUTE B=...
Ø51	CADD	A	+A
Ø52	CSUB	A	-A
Ø53	CMUL	A	*A
Ø54	CDIV	A	/A
Ø55	CEXP	A	**A
Ø56			
Ø57	CEND	-	END OF COMPUTE
Ø6Ø	ACCEPT	A,...[,B]	ACCEPT A,... [FROM B]
Ø61	DISPLY	A,...[,B]	DISPLAY A,... [UPON B]
Ø62	OPEN	A	OPEN A
Ø63	CLOSE	A	CLOSE A
Ø64	READ	A	READ A
Ø65	WRITE	A, [B]	WRITE A [ADVANCING B]
Ø66			
Ø67	SEEK	A	SEEK A
Ø7Ø	LPAREN	-	LEFT PARENTHESIS
Ø71	RPAREN	-	RIGHT PARENTHESIS
Ø72	EXP	-	START EXPRESSION
Ø73	ENDEXP	-	END EXPRESSION
Ø74	JUMPTO	-	GENERATED CONTROL TRANSFER
Ø75	ERAUSE	-	ERROR USE PROCEDURE
Ø76	CLREOP	-	CLEAR EOPTAB
1ØØ	SECNAM	A	SECTION-NAME-A
1Ø1	PARNAM	A	PARAGRAPH-NAME-A
1Ø2	TAGNAM	A	SPECIAL TAG (%NNNNN)
1Ø3	SENAM	A	REFERENCE POINT FOR SENTENCES
1Ø4	ENDSEC	-	END OF SECTION
1Ø5	YECCH	-	IGNORE ALL PRECEDING OPERANDS
11Ø	SORT	A	SORT A ...
111	KEY	A	KEY A
112	INPROC	A, [B]	INPUT PROCEDURE IS A [THRU B]
113	OUTPRC	A, [B]	OUTPUT PROCEDURE IS A [THRU B]
114	GIVING	A	GIVING A
115	USING	A	USING A
116	ENDSRT	-	END OF SORT STATEMENT
117			
12Ø	RELEAS	A, [B]	RELEASE A [FROM B]
121	RETURN	A, [B]	RETURN A [INTO B]
.			(ERROR)
377	ENDIT	-	END OF SOURCE

(GEN CONT'D)

.6.5 FLAGS USED WITH THE OPERATORS

ACCEPT:

BIT 9	"FROM" OPTION
BITS 1Ø-15	NOT USED

CADD,CSUB,CMUL,CDIV,CEXP:

BIT 9	"UNARY MINUS"--NEGATE OPERAND
BITS 1Ø-15	NOT USED

CEND:

BIT 9	ROUNDED
BITS 1Ø-15	NOT USED

CLOSE:

BIT 9	FILE (Ø) OR REEL (1)
BIT 1Ø	WITH LOCK
BIT 11	NO REWIND
BITS 12-15	NOT USED

DISPLY:

BIT 9	"UPON" OPTION
BITS 1Ø-15	NOT USED

DIV:

BIT 9	"INTO" (SWAP OPERANDS)

DIVBY:

BIT 9	"INTO" (SWAP OPERANDS)

ENDIF:

BIT 9	END "SPIF"

ENTER:

BIT 9	MACRO
BIT 1Ø	FORTRAN-IV
BIT 11	USING
BITS 12-15	NOT USED

EXAM:

BIT 9	"LEADING"
BIT 1Ø	"FIRST"
BIT 11	"UNTIL FIRST"
BIT 12	"REPLACING"
BIT 13	"TALLYING"
BIT 14-15	NOT USED

(2.6.5 GEN OPERATOR FLAGS CONT'D)

GO:

BIT 9	SPECIAL GENERATED GO FOR END OF SECTION
BIT 1Ø	SAME AS BIT 9, EXCEPT FOR PHYSICALLY LAST SECTION

IF:

BIT 9	"LESS"
BIT 1Ø	"GREATER"
BIT 11	"EQUAL"
BIT 12-13	NOT USED
BIT 14	TAG IN LH OF WORD 2 IS FOR FALSE PATH
BIT 15	"NOT"

IFC:

BIT 9	"ON" TEST FOR SWITCH
BIT 1Ø	"OFF" TEST FOR SWITCH
BIT 11-13	NOT USED
BIT 14	TAG IN LH OF WORD 2 IS FOR FALSE PATH
BIT 15	"NOT"

IFT:

BIT 9	"NUMERIC"
BIT 1Ø	"ALPHABETIC"
BIT 11	"POSITIVE"
BIT 12	"NEGATIVE"
BIT 13	"ZERO"
BIT 14	TAG IN LH OF WORD 2 IS FOR FALSE PATH
BIT 15	"NOT"

KEY:

BIT 9	DESCENDING (Ø) OR ASCENDING (1)
BITS 1Ø-15	NOT USED

OPEN:

BIT 9	OUTPUT
BIT 1Ø	INPUT
BIT 11	NO REWIND
BITS 12-15	NOT USED

READ:

BIT 9	"INTO"
BITS 1Ø-15	NOT USED

RELEAS: SEE "WRITE"

REMAIN:
SEE "RESULT"

RESULT:

BIT 9	NOT USED
BIT 1Ø	"SIZE ERROR"
BITS 11	NOT USED
BIT 12	"CORRESPONDING"
BITS 13-15	NOT USED

(2.6.5 GEN OPERATOR FLAGS CONT'D)

```
      RETURN: SEE "READ"

      SPIF:
        BIT 9            AT END
        BIT 1Ø           SIZE ERROR
        BIT 11           INVALID KEY
        BIT 12           SIZE ERROR FOR CORRESPONDING
        BITS 13-15       NOT USED

      STASH:
        BITS 9-15        CONTAIN A NUMBER USED TO LATER REFERENCE
                         THE TEMPORARY (SEE 2.6.2., BITS 18-2Ø,21-35)
      STOP:
        BIT 9            "STOP RUN"
        BITS 1Ø-15       NOT USED

      USE:
        BITS 9-1Ø        CODE FOR TYPE:
                         ØØ ALL INPUT
                         Ø1 ALL OUTPUT
                         1Ø ALL INPUT-OUTPUT
                         11 ERROR
        BIT 11           BEFORE (Ø) OR AFTER (1)
        BIT 12           BEGINNING
        BIT 13           ENDING
        BIT 14           REEL/UNIT
        BIT 15           FILE

      WRITE:
        BIT 9            ADVANCING
        BIT 1Ø           BEFORE (Ø) OR AFTER (1)
        BIT 11           "FROM"
        BITS 12-15       NOT USED
```

2.7 CPY - THIS FILE CONTAINS A COPY OF THE SOURCE FILE, WITH LINE NUMBERS ASSIGNED, AND SOME EDITING DONE.

THERE IS ONE WORD FOR EACH LINE WHICH CONTAINS:

BITS Ø-6 THE PRINTER CONTROL CHARACTER FOR THE PRECEDING LINE (FORM-FEED OR LINE-FEED)

BIT 7 ALWAYS ZERO

BITS 8-2Ø THE ASSIGNED LINE NUMBER FOR THIS LINE

BITS 21-34 THE FIRST TWO CHARACTERS OF THE SOURCE LINE
BIT 35 ALWAYS 1 TO IDENTIFY THIS AS THE HEADER WORD

THE REMAINING WORDS FOR EACH LINE CONTAIN THE ASCII CHARACTERS COPIED FROM THE SOURCE FILE.

2.8 AS1,AS2,AS3 - THE OUTPUT OF THE CODE GENERATOR, PHASE E ARE FILES CONTAINING DIRECTIONS FOR THE ASSEMBLY PHASE. EACH ENTRY IN THE FILE CONSISTS OF A HEADER WORD FOLLOWED BY (USUALLY) MORE DEFINITIVE DATA.

2.8.1 ADDRESSES - THE HEADER DATA WORDS MAY CONTAIN AN ADDRESS. THAT ADDRESS HAS A TYPE-CODE IN ITS FIRST 3 BITS, FOLLOWED BY AN ADDRESS RELATIVE TO SOME TABLE. THE TYPE-CODES ARE:

Ø	THE REMAINDER OF THE ADDRESS IS A CONSTANT
1	USE THE DATA-NAME TABLE
2	USE THE PROCEDURE-NAME TABLE
3	USE THE EXTERNAL-NAME TABLE
4	USE THE FILE-NAME TABLE
5	USE THE GENERATED-TAG TABLE
6	THE INCREMENT (SEE 2.8.2) IS A CONSTANT
7	MISCELLANEOUS (SEE 2.8.2).

2.8.2 ADDRESS INCREMENT - OCCASIONALLY THE ADDRESS IS MODIFIED BY SOME INCREMENT. THE INCREMENT HAS A TYPE-CODE IN ITS FIRST 3 BITS, FOLLOWED BY A CONSTANT AMOUNT IN INCREMENT

THE VALUES FOR THE TYPE CODE ARE:

Ø	ADD CONSTANT TO THE ADDRESS
1	ADD CONSTANT TO BASE OF PARAMETERS IN IMPURE AREA
2	NOT USED
3	ADD CONSTANT TO THE BASE OF THE LITERAL POOL
4	REFERENCES GOTO..
5	ADD CONSTANT TO CURRENT LOCATION
6	ADD CONSTANT TO BASE OF RUN-TIME IMPURE AREA
7	ADD CONSTANT TO ALTER TABLE FOR SEGMENTS > 49.

2.8.3 INSTRUCTION

WORD 1:

BIT Ø	ALWAYS Ø TO IDENTIFY THIS AS A PDP-1Ø INSTRUCTION.
BIT 1	IF THERE IS AN INCREMENT WORD FOLLOWING THIS, THIS BIT IS A 1.
BITS 2-8	OPERATION CODE (SEE 2.8.8)
BITS 9-12	ACCUMULATOR FIELD FOR THE INSTRUCTION
BIT 13	INDIRECT BIT FOR THE INSTRUCTION
BIT 14-17	INDEX FIELD FOR THE INSTRUCTION
BITS 18-35	THE ADDRESS (SEE 2.8.1)

WORD 2:

EXISTS ONLY IF BIT 1 OF WORD 1 WAS A 1. IT HAS ZERO IN THE LEFT-HALF, AND AN INCREMENT (SEE 2.8.2) IN THE RIGHT HALF.

(2.8 AS1, AS2, AS3 CONT'D)

2.8.4 BYTE POINTER

WORD 1:

BITS Ø-2	ALWAYS 4
BITS 3-17	NOT USED
BITS 18-35	ADDRESS (SEE 2.8.1)

WORD 2:

BITS Ø-17	THE LEFT-HALF OF THE POINTER WORD, AS GENERATED BY THE MACRO-1Ø "POINT" PSEUDO-OP.
BITS 18-35	INCREMENT (SEE 2.8.2)

2.8.5 XWD

WORD 1:

BITS Ø-2	ALWAYS 5
BITS 3-17	NOT USED
BITS 18-35	NUMBER OF FOLLOWING 2-WORD ENTRIES

THE HEADER WORD IS FOLLOWED BY ONE OR MORE 2-WORD ENTRIES:

FIRST WORD:

BITS Ø-17	ADDRESS INCREMENT FOR LEFT-HALF OF THE XWD
BITS 18-35	ADDRESS FOR LEFT-HALF OF XWD

SECOND WORD:

BITS Ø-17	ADDRESS INCREMENT FOR RIGHT HALF OF XWD
BITS 18-35	ADDRESS FOR RIGHT HALF OF XWD

2.8.6 CONSTANT

WORD 1:

BITS Ø-2	ALWAYS 6
BIT 3	ASCII
BIT 4	SIXBIT
BIT 5	DECIMAL (ONE-WORD)
BIT 6	DECIMAL (TWO-WORDS)
BIT 7	FLOATING POINT
BIT 8	OCTAL
BITS 9-12	NOT USED
BITS 13-17	NUMBER OF CHARACTERS IN FLOATING POINT MANTISSA
BITS 18-35	NUMBER OF WORDS CONTAINING THE CONSTANT

FOLLOWING THE HEADER WORD IS THE VALUE OF THE CONSTANT, IN AS MANY WORDS AS ARE NECESSARY.
FLOATING POINT CONSTANTS ARE TWO WORDS APIECE:
THE FIRST WORD IS THE TENS EXPONENT, IN BINARY;
THE SECOND WORD IS THE MANTISSA, 4 BITS PER DIGIT.

2.8.7 MISCELLANEOUS

WORD 1:

BITS Ø-2	ALWAYS 7
BIT 3	PARAGRAPH OR SECTION-NAME (FOR LISTING PURPOSES)
BIT 4	SPECIAL TAG (FOR LISTING PURPOSES)
BIT 5	"RELOC" PSEUDO-OP
BITS 6-16	NOT USED
BIT 17	INCREMENT WORD FOLLOWS
BITS 18-35	ADDRESS (SEE 2.8.1)

WORD 2: PRESENT ONLY IF BIT 17 OF WORD 1 IS A 1.

BITS Ø-17	NOT USED
BITS 18-35	ADDRESS INCREMENT (SEE 2.8.2)

2.8.8 INSTRUCTION CODES

FOR A LIST OF INSTRUCTION CODES, SEE ROUTINE "CMNGEN".

TABLE STRUCTURE FOR PDP-1Ø COBOL COMPILER

THE INFORMATION ON THIS MEMORANDUM IS SUBJECT TO CHANGE WITHOUT NOTICE AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT CORPORATION.

1.Ø GENERAL

ALL OF THE MAJOR TABLES USED BY THE COBOL COMPILER ARE DESCRIBED IN THE FOLLOWING PARAGRAPHS. THESE TABLES RESIDE IN THE IMPURE AREA OF CORE DURING COMPILATION.

EACH TABLE IS ALLOCATED SOME NOMINAL AMOUNT OF CORE DURING THE INITIALIZATION PHASE. IF MORE CORE IS NEEDED FOR A SPECIFIC TABLE DURING COMPILATION, THE CORE UUO WILL BE CALLED, ALL TABLES IN HIGHER LOCATIONS WILL BE MOVED UP, AND THE EXTRA CORE WILL BE CLEARED.

THERE IS A LOCATION POINTER IN THE IMPURE AREA FOR EACH TABLE. THIS POINTER HAS THE NEGATIVE OF THE SIZE OF THE TABLE IN ITS LEFT HALF, AND THE STARTING ADDRESS OF THE TABLE IN ITS RIGHT-HALF. THIS POINTER WORD ALLOWS THE TABLES TO BE MOVED AROUND AS THE CORE ALLOCATION ALGORITHM SEES FIT.

1.1 TABLE-LINKS

CERTAIN ENTRIES IN THE TABLE ARE CALLED "TABLE-LINKS". THESE ARE 18-BIT FIELDS CONTAINING:

BITS Ø-2	TABLE TYPE -	Ø	FILTAB
		1	DATAB
		2	CONTAB
		4	PROTAB
		5	EXTAB
		7	MNETAB

BITS 3-17 ADDRESS OF A TABLE ENTRY, RELATIVE
TO THE BEGINNING OF THAT TABLE.

2. NAMTAB -- NAME TABLE

2.1 USED BY: PHASES B, C, D, F, C
ENTRY SIZE: 3 OR 6 WORDS
INITIAL CORE
ALLOCATION: 2Ø48 WORDS (APPROX. 6ØØ ENTRIES)

2.2 CONTENTS

THIS TABLE CONTAINS INFORMATION ABOUT ALL WORDS FOUND IN THE SOURCE-FILE. IT DOES NOT INCLUDE ENTRIES FOR LITERALS OR PICTURES.

2.3 SEARCH TECHNIQUE

A DESCRIPTION OF THE SEARCH TECHNIQUE CAN BE FOUND IN THE PROGRAM "TRYNAM".

2.4 DETAILED DESCRIPTION

WORD 1:

BITS Ø-1 ALWAYS ØØ TO IDENTIFY THE START OF AN ENTRY.

BIT 2 SET TO A 1 IF THIS IS A COBOL RESERVED WORD.

BITS 3-17 IF THIS IS A RESERVED WORD, THIS FIELD C
ONTAINS
A VALUE (SEE PROGRAM "RESVWD" FOR A COMPLETE LIST OF VAL
UES).

BITS 18-2Ø IF NOT A RESERVED WORD, THIS DETERMINES THE TYPE OF ITEM (SEE 1.1).

WORDS 2-N: THE WORD, IN SIXBIT, OVER AS MANY WORDS AS NECESSARY. HYPHENS ARE REPRESENTED BY COLONS, PERIODS BY SEMI-COLONS. THE WORD IS TERMINATED BY THE FIRST CHARACTER HAVING ZEROES IN THE HI-ORDER 2 BITS.

Pages 4 and 5

have been

deleted

Pages 4 and 5

have been

deleted

3. FILTAB - FILE TABLE

3.1 USED BY: PHASES B, C, D, E, F, G
ENTRY SIZE: 15+ WORDS
INITIAL CORE
ALLOCATION: 15Ø WORDS (APPROX. 1Ø ENTRIES)

3.2 CONTENTS

THIS TABLE CONTAINS INFORMATION ABOUT THE FILES SELECTED BY THE SOURCE PROGRAM.

3.3 DETAILED DESCRIPTION

WORD 1:

BITS Ø-2	ALWAYS Ø TO IDENTIFY THE TABLE.
BITS 3-17	POINTER TO NAMTAB ENTRY FOR THIS FILE NAME.
BITS 18-35	TABLE-LINK TO AN ITEM HAVING THE SAME NAME AS THIS FILE.

WORD 2:

BIT Ø	MULTIPLE REEL/UNIT
BITS 1-17	COUNT SPECIFIED IN RERUN
BITS 18-35	LOCATION OF OBJECT-TIME FILE TABLE

WORD 3:

BITS Ø-15	THE SIZE OF THE BUFFER FOR THIS FILE, IN CHARACTERS.
BITS 16-28	THE ASSIGNED SOURCE LINE NUMBER FOR THE "SELECT" FOR THIS FILE. THIS IS USED WHEN DIAGNOSTICS ARE PUT OUT.
BITS 29-35	THE POSITION WITHIN THE SOURCE LINE CONTAINING THE FIRST CHARACTER OF THE FILE NAME. THIS IS USED IN CONJUNCTION WITH BITS 16-28 WHEN DIAGNOSTICS ARE PUT OUT.

(3.3 FILTAB CONT'D)

WORD 4:

BITS Ø-11 SIZE OF A NON-STANDARD LABEL RECORD, IN CHARACTERS.

BITS 12-17 NUMBER OF DEVICES ASSOCIATED WITH THIS FILE.

BITS 18-35 TABLE-LINK TO VALTAB FOR FIRST DEVICE NAME.

WORD 5:

BITS Ø-4 NUMBER OF "FILE-LIMITS" CLAUSES.

BITS 5-11 BLOCKING FACTOR, IN RECORDS

BITS 12-17 FILE POSITION IN A MULTI-FILE REEL.

BITS 18-35 TABLE-LINK TO THE NEXT FILTAB ENTRY.

WORD 6:

BITS Ø-1 RECORDING MODE ON EXTERNAL MEDIA:
ØØ - SIXBIT
Ø1 - BINARY
1Ø - ASCII
11 - NOT YET DECLARED

BITS 2-3 LABELS: ØØ OMITTED*
Ø1 STANDARD
1Ø NON-STANDARD
11 NOT YET DECLARED

BIT 4 THERE ARE INPUT "OPENS" IN PROCEDURE DIVISION.

BIT 5 THERE ARE OUTPUT "OPENS" IN PRECEDURE DIVISION.

BIT 6 THERE ARE I/O "OPENS" IN PROCEDURE DIVISION.

BIT 7 "WRITE ADVANCING" WAS SEEN IN PROCEDURE DIVISION

BIT 8 DEFINED IN AN SD

(3.3 FILTAB CONT'D)

```
                BIT 9              DATA RECORDS ARE VARIABLE
                                   LENGTH.

                BIT 1Ø             RERUN ON END-OF-REEL

                BIT 11             RERUN ON COUNT

                BIT 12             FD OR SD IS DEFINED

                BIT 13             OPTIONAL FILE

                BITS 14-15         RECORDING MODE IN CORE:
                                   ØØ - SIXBIT
                                   Ø1 - BINARY
                                   1Ø - ASCII
                                   11 - NOT YET DECLARED

                BITS 16-17         ACCESS MODE:
                                   ØØ - SEQUENTIAL
                                   Ø1 - RANDOM
                                   1Ø - NOT USED
                                   11 - NOT YET DECLARED

                BITS 18-35         TABLE-LINK TO THE "ACTUAL KEY"

         WORD 7:
                BITS Ø-5           NUMBER OF BUFFERS

                BITS 6-17          MAXIMUM DATA RECORD SIZE IN
                                   CHARACTERS.

                BITS 18-35         TABLE-LINK TO FIRST DATA RECORD.

         WORD 8: LH                SIZE OF LABEL RECORD, IN CHARACTERS
                                   (IF NON-STANDARD LABELS).
                 RH                TABLE-LINK TO ANOTHER FILE TABLE WHICH
                                   IS IN A MULTI-FILE CLAUSE WITH THIS FILE


         WORD 9:  LH               TABLE-LINK TO THE VALUE-OF-IDEN-
                                     TIFICATION.
                  RH               TABLE-LINK TO THE VALUE-OF-DATE-
                                     WRITTEN.

         WORD 1Ø: LH               TABLE-LINK TO A FILE USING THE SAME BUFF
ER AREA
                  RH               TABLE-LINK TO "ERROR USE"

         WORD 11: LH               TABLE-LINK TO "BEFORE BEGINNING
                                     REEL USE"
                  RH               TABLE-LINK TO "BEFORE BEGINNING
                                     FILE USE".
```

(3.3 FILTAB CONT'D)

WORD 12:	LH	TABLE-LINK TO "AFTER BEGINNING REEL USE".
	RH	TABLE-LINK TO "AFTER BEGINNING FILE USE".
WORD 13:	LH	TABLE-LINK TO "BEFORE ENDING REEL USE".
	RH	TABLE-LINK TO "BEFORE ENDING FILE USE".
WORD 14:	LH	TABLE-LINK TO "AFTER ENDING REEL USE".
	RH	TABLE-LINK TO "AFTER ENDING FILE USE".
WORD 15:	LH	TABLE-LINK TO A FILE SHARING THE SAME RECORD AREA.
	RH	TABLE-LINK TO DATAB ENTRY FOR LABEL.
WORD 16:	BIT Ø	DATA RECORDS CLAUSE APPEARED IN FD OR SD
	BIT 1	RH CONTAINS RECORD AREA ADDRESS
	BITS 2-17	NOT USED
	BITS 18-35	BASE ADDRESS OF RECORD AREA

WORDS 17-N:

THESE WORDS ARE PRESENT ONLY FOR RAC DEVICES. THE LEFT-HALF OF EACH ENTRY IS A TABLE-LINK TO A "LOW-LIMIT", THE RIGHT HALF IS A TABLE-LINK TO A "HIGH-LIMIT", AS DESCRIBED IN THE "FILE-LIMITS" CLAUSE FOR THIS FILE.

4. DATAB - DATA DESCRIPTOR TABLE

4.1 USED BY: PHASES C,D,E,F,G
ENTRY SIZE: 5-1Ø WORDS
INITIAL CORE
ALLOCATION: 12ØØ WORDS (APPROX. 2ØØ ENTRIES)

4.2 CONTENTS

DATAB CONTAINS INFORMATION ABOUT EACH DATA DIVISION ITEM DEFINED BY THE USER, OTHER THAN CONDITION-NAMES.

4.3 DETAILED DESCRIPTION

WORD1:

BITS Ø-2 ALWAYS 1 TO IDENTIFY THE TABLE.

BITS 3-17 POINTER TO NAMTAB ENTRY FOR THIS ITEM

BITS 18-35 TABLE-LINK TO AN ITEM HAVING THE SAME NAME AS THIS ITEM.

WORD 2: LH TABLE-LINK TO ANY VALUE

RH DURING PHASE C, THIS IS A TABLE LINK TO THE REDEFINED ITEM (SEE WORD 5, BIT 2Ø)

IN LATER PHASES, THIS IS THE ASSIGNED CO

RE

LOCATION FOR THIS ITEM.

WORD3: LH TABLE-LINK TO THE GROUP ITEM CONTAINING THIS ITEM ("FATHER"), OR TABLE-LINK TO NEXT ITEM WITH THE SAME LEVEL NUMBER ("BROTHER"). SEE WORD 5, BIT 8.

RH TABLE-LINK TO FIRST ITEM OF A HIGhER LEVEL NUMBER ("SON").

1.3 DATAB CONT'D)

WORD4:	BITS Ø-5	LEVEL NUMBER
	BITS 6-11	BYTE-RESIDUE - IF THIS IS A DISPLAY ITEM, NUMBER USED TO BUILD A BYTE POINTER TO THIS ITEM
	BITS 12-35	NOT USED
WORD 5:	BITS Ø-1	CLASS: ØØ - ALPHANUMERIC Ø1 - ALPHABETIC 1Ø - NUMERIC 11 - NOT YET DECLARED
	BIT 2	SYNCHRONIZED LEFT
	BIT 3	SYNCHRONIZED RIGHT
	BIT 4	SIGNED
	BIT 5	BLANK WHEN ZERO
	BIT 6	THIS ITEM MUST BE SUBSCRIPTED OR INDEXED
	BIT 7	EDITED ITEM
	BIT 8	BROTHER (Ø) OR FATHER (1) LINK IN WORD 3.
	BIT 9	ITEM IS DEFINED
	BIT 1Ø	THIS IS USED AS AN "ACTUAL KEY"
	BIT 11	THIS IS A FILE-LIMIT FOR A RAC DEVICE
	BIT 12	THIS IS A "VALUE OF ID"
	BIT 13	THIS IS A "VALUE OF DATE-WRITTEN"

(4.3 DATAB CONT'D)

	BIT 14	JUSTIFIED LEFT (Ø) OR RIGHT (1)
	BITS 15-17	USAGE: ØØØ NOT YET DECLARED ØØ1 DISPLAY-6 Ø1Ø DISPLAY-7 1ØØ 1-WORD COMP 1Ø1 2-WORD COMP 11Ø COMP-1 111 INDEX
	BIT 18	ERROR DETECTED IN DATA DIVISION
	BIT 19	THIS IS AN INDEX-NAME
	BIT 2Ø	THIS ITEM REDEFINES ANOTHER (SEE WORD 2, BITS 18-35)
	BIT 21	PICTURE SEEN
	BIT 22	ITEM DEFINED IN FILE SECTION
	BIT 23	THIS APPEARS IN A DATA RECORDS CLAUSE
	BIT 24	THIS APPEARS IN A LABEL RECORDS CLAUSE
	BIT 25	THERE ARE SYNCS AT LOWER LEVEL
	BIT 26	PICTURE WORDS ALLOCATED
	BITS 27-29	NOT USED
	BIT 3Ø	DECIMAL POINT IS TO RIGHT OF WORD BOUNDARY (E.G. PICTURE 9PPPV)
	BITS 31-35	NUMBER OF DECIMAL PLACES
WORD 6:		
	BITS Ø-17	EXTERNAL SIZE
	BITS 18-35	INTERNAL SIZE
WORD 7:		
	BITS Ø-14	NUMBER OF OCCURRENCES
	BIT 15	OCCURS CLAUSE IS AT THIS ITEM
	BITS 16-28	LINE NUMBER IN SOURCE
	BITS 29-35	CHARACTER POSITION IN SOURCE

(4.3 DATAB CONT'D)

WORD 8: LH LINK TO NEXT HIGHER OCCURENCE LEVEL
RH LINK TO DEPENDING ITEM

WORD 9: OPTIONAL--USED ONLY IF ITEM IS SUBSCRIPTED OR EDITED.

BITS Ø-11 EXTERNAL SIZE OF THE ITEM, IF THIS IS THE FIRST LEVEL OF "OCCURS". EXTERNAL SIZE OF THE ITEM WITH NEXT LOWER LEVEL NUMBER, IF THIS IS NOT THE FIRST LEVEL OF "OCCURS".

BITS 12-23 IF THIS IS THE SECOND OR THIRD LEVEL OF OCCURS, THIS FIELD CONTAINS THE NUMBER OF OCCURANCES OF THIS ITEM.

BITS 24-35 IF THIS IS THE THIRD LEVEL OF OCCURS, THIS FIELD CONTAINS THE NUMBER OF OCCURRENCE AT THE SECOND LEVEL.

WORDS 1Ø-12: OPTIONAL - USED ONLY IF THE ITEM IS EDITED.

BITS Ø-5 THE PICTURE SIGN CHARACTER, IN SIXBIT

BITS 6-11 THE PICTURE FLOAT OR SUPPRESSION CHARACTER, IN SIXBIT.

THE REMAINDER OF THE FIELD IS COMPOSED OF 4-BIT BYTES TO BE USED BY THE PICTURE EDITING ROUTINE AT OBJECT TIME. THE VALUES FOR THESE BYTES ARE:

ØØ - INSERT AN ALPHA CHARACTER (X OR A)
Ø1 - INSERT A NUMERIC CHARACTER (9)
Ø2 - SUPPRESS (Z OR *)
Ø3 - FLOAT ($,+,-)
Ø4 - INSERT A COMMA
Ø5 - INSERT A SPACE (B)
Ø6 - INSERT A ZERO (Ø)
Ø7 - INSERT A CURRENCY SIGN
1Ø - INSERT A SIGN (+,-)
11 - INSERT A DECIMAL POINT
12 - INSERT "CR"
13 - INSERT "DB"
14 - UNUSED
15 - UNUSED
16 - UNUSED
17 - TERMINATE EDITING

5. CONTAB - CONDITION-NAME TABLE

5.1 USED BY: PHASES C, D, E
ENTRY SIZE: VARIABLE
INITIAL CORE
ALLOCATION: 5Ø WORDS

5.2 CONTENTS

CONTAB CONTAINS INFORMATION ABOUT 88-LEVEL ITEMS IN THE DATA DIVISION

5.3 DETAILED DESCRIPTION

WORD 1:

BITS Ø-2 ALWAYS 2 TO IDENTIFY THE TABLE

BITS 3-17 POINTER TO NAMTAB ENTRY FOR THIS ITEM

BITS 18-35 TABLE-LINK TO ANOTHER ITEM WITH THE SAME NAME

WORD 2:

BITS Ø-17 TABLE-LINK TO THE DATAB ITEM FOR WHICH THIS ITEM IS A CONDITION-NAME

BITS 18-35 THE NUMBER OF LITERAL ENTRIES

THE REMAINING WORDS CONTAIN THE VALUE OR VALUES FOR THE CONDITIONS. THE FIRST WORD OF EACH VALUE ENTRY HAS:

BIT Ø A 1 INDICATES THAT THIS VALUE IS THE FIRST OF A RANGE ("VALUE IS A THRU B").

BIT 1 THE VALUE IS A FIGURATIVE CONSTANT

BIT 2 "ALL"

5.3 CONTAB CONT'D)

BITS 3-17

IF THIS IS NOT A FIGURATIVE CONSTANT, THIS IS A TAG NUMBER. IF THIS IS A FIGURATIVE CONSTANT THE BITS HAVE THE FOLLOWING MEANINGS:
BIT 3 "SPACE", "SPACES"
BIT 4 "ZERO", "ZEROES", "ZEROS"
BIT 5 "QUOTE", "QUOTES"
BIT 6 "HIGH-VALUE", "HIGH-VALUES"
BIT 7 " LOW-VALUE", "LOW-VALUES"
BIT 8-17 NOT USED

BITS 18-35

SAME AS Ø-17 FOR SECOND PART OF "THRU"

6. PROTAB - PROCEDURE-NAME DESCRIPTOR TABLE

6.1 USED BY: PHASES D,E,F,G
ENTRY SIZE: 4 WORDS
INITIAL CORE
ALLOCATION: 4ØØ WORDS (1ØØ ENTRIES)

6.2 CONTENTS

PROTAB CONTAINS INFORMATION ABOUT PARAGRAPH AND SECTION NAMES.

6.3 DETAILED DESCRIPTION

WORD 1:

BITS Ø-2 ALWAYS 4 TO IDENTIFY THE TABLE

BITS 3-17 POINTER TO THE NAMTAB ENTRY FOR THIS ITEM

BITS 18-35 TABLE-LINK TO ANOTHER ITEM WITH THE SAME NAME.

WORD 2:

BITS Ø-17 IF THIS IS A PARAGRAPH-NAME, THIS FIELD IS A TABLE-LINK TO THE PROTAB ENTRY CONTAINING THE SECTION-NAME FOR THE SECTION CONTAINING THIS PARAGRAPH.
IF THIS IS A SECTION NAME, THIS FIELD CONTAINS THE WORD NUMBER IN GENFIL WHICH CONTAINS THE SECNAM OPERATOR FOR THE NEXT SECTION; OR ZERO IF THIS IS THE LAST SECTION.

BITS 18-35 THE LOCATION ASSIGNED TO THE FIRST INSTRUCTION GENERATED FOR THIS PARAGRAPH OR SECTION.

(6.3 PROTAB CONT'D)

WORD 3:

BITS Ø-17	PARAMETER ADDRESS FOR ALTER WORD
BITS 18-24	THE SECTION PRIORITY NUMBER (Ø IF RESIDENT)
BIT 25	ITEM IS A SECTION (Ø) OR PARAGRAPH (1) NAME
BIT 26	EXIT REQUIRED (REFERENCE IN "THRU" CLAUSE OF A "PERFORM")
BIT 27	ITEM IS DEFINED
BIT 28	PARAGRAPH IS ALTERABLE
BIT 29	AN OBJECT OF AN ALTER IS IN THE CURRENT SEGMENT
BIT 3Ø	AN OBJECT OF AN ALTER IS IN ANOTHER SEGMENT
BIT 31	REFERENCED IN THE DECLARATIVES
BIT 32	ITEM IS IN THE DECLARATIVES
BIT 33	THIS IS MULTIPLY DEFINED
BIT 34	THIS TERMINATES WITH AN UNCONDITIONAL TRANSFER
BIT 35	THIS SECTION HAS APPEARED AS A SECNAM ARGUMENT BEFORE.

WORD 4:

LH	RELATIVE ADDRESS OF AN EXIT WORD
RH	THIS IS THE RELATIVE ADDRESS OF AN ENTRY IN FLOTAB.

7. EXTAB - EXTERNAL-NAME TABLE

7.1

USED BY:	PHASES D, E, F, G
ENTRY SIZE:	2 WORDS
INITIAL CORE ALLOCATION:	4Ø WORDS (2Ø ENTRIES)

7.2 CONTENTS

EXTAB CONTAINS INFORMATION ABOUT NAMES OF EXTERNAL ROUTINES.

7.3 DETAILED DESCRIPTION

WORD 1:

BITS Ø-2	ALWAYS 5 TO IDENTIFY THE TABLE
BITS 3-17	POINTER TO NAMTAB ENTRY FOR THIS NAME
BITS 18-35	TABLE-LINK TO ANOTHER ITEM WITH THE SAME NAME

WORD 2:

BIT Ø	A 1-BIT IF THE ENTRY IS REFERENCED BY A NON-RESIDENT SEGMENT.
BIT 1	OP-SYS (Ø) OR USER (1) NAME
BIT 2	THIS IS THE PROGRAM-ID.
BITS 3-17	NOT USED
BITS 18-35	LOCATION OF PREVIOUS REFERENCE TO THIS ROUTINE (USED AT ASSEMBLY TIME)

8. MNETAB - MNEMONIC-NAME TABLE

8.1 USED BY: PHASES B,D,E
ENTRY SIZE: 2 WORDS
INITIAL CORE
ALLOCATION: 4Ø WORDS (2Ø ENTRIES)

8.2 CONTENTS CONTAINS INFORMATION ABOUT MNEMONIC-NAMES FOUND IN THE SPECIAL-NAMES PARAGRAPH IN THE ENVIRONMENT DIVISION.

8.3 DETAILED DESCRIPTION

WORD 1:

BITS Ø-2	ALWAYS 7 TO IDENTIFY THE TABLE
BITS 3-17	POINTER TO NAMTAB ENTRY FOR THIS ITEM
BITS 18-35	TABLE-LINK TO ANOTHER ITEM WITH THE SAME NAME.

WORD 2:

BIT Ø	"SWITCH"
BIT 1	"SWITCH ON STATUS"
BIT 2	"SWITCH OFF STATUS"
BIT 3	"CONSOLE"
BIT 4	"CHANNEL"
BITS 5-29	NOT USED
BITS 30-35	IF BITS Ø, 1 OR 2 ARE A 1, THIS IS A SWITCH NO. IF BIT 4 IS A 1, THIS IS A CHANNEL NO.

9. RESTAB - RESULT TABLE

9.1

USED BY:	PHASE E
ENTRY SIZE:	2 WORDS
INITIAL CORE ALLOCATION:	2Ø WORDS (1Ø ENTRIES)

9.2 CONTENTS

RESTAB CONTAINS INFORMATION ABOUT "RESULT" OPERANDS AND ENABLES PHASE E TO GENERATE BETTER CODE.

9.3 DETAILED DESCRIPTION:

WORD 1:

BIT Ø	RESULT IS TO BE ROUNDED
BITS 1-17	NOT USED
BITS 18-35	ABSOLUTE ADDRESS OF AN ENTRY IN EOPTAB

WORD 2:

BITS Ø-17	NUMBER OF INTEGRAL PLACES IN ITEM
BITS 18-35	NUMBER OF DECIMAL PLACES IN ITEM

1Ø. VALTAB - VALUE TABLE

1Ø.1 USED BY: PHASES B,C
ENTRY SIZE: VARIABLE
INITIAL CORE
ALLOCATION: 1ØØ WORDS

1Ø.2 CONTENTS

VALTAB HOLDS THE LITERALS FOUND IN THE "VALUE" CLAUSE IN THE DATA DIVISION WHILE A RECORD IS BEING PROCESSED, AND OTHER MISCELLANEOUS LITERALS WHILE THERE ARE NEEDED (E.G. FILE-LIMITS, VALUE OF ID). THE TABLE IS CLEARED WHEN THE INFORMATION CAN BE WRITTEN OUT ONTO THE ASSEMBLY INPUT FILE.

1Ø.3 DETAILED DESCRIPTION

EACH VALUE IS PLACED IN ONE OR MORE WORDS. THE FIRST WORD HAS THE NUMBER OF CHARACTERS IN BITS Ø-6; THE REMAINDER OF THAT WORD, AND THE FOLLOWING WORDS, CONTAIN AN ASCII STRING.

11. LITAB - LITERAL TABLE

11.1 USED BY: PHASE E
ENTRY SIZE: VARIABLE
INITIAL CORE
ALLOCATION: 1ØØ WORDS

11.2 CONTENTS

LITAB CONTAINS INFORMATION ABOUT LITERALS GENERATED BY THE CODE GENERATION PHASE. AT EACH SEGMENT BREAK, THE INFORMATION IS WRITTEN OUT INTO AN ASSEMBLY INPUT FILE, AND THE CONTENTS OF THE TABLE ARE FLUSHED.

11.3 DETAILED DESCRIPTION

EACH ENTRY CONSISTS OF A HEADER WORD FOLLOWED BY WORDS CONTAINING THE LITERAL VALUE.

IN PHASE C, THE HEADER WORD CONTAINS:

BITS Ø-5	NOT USED
BIT 6	NON-SIXBIT
BIT 7	ALL
BIT 8	NUMERIC
BIT 1Ø	NUMERIC LITERAL HAS AN IMBEDDED DECIMAL POINT
BITS 11-17	NUMBER OF CHARACTERS IN THE LITERAL
BITS 18-35	NUMBER OF WORDS CONTAING THE LITERAL

(11.3 CONT'D)

IN PHASE E, THE HEADER WORD CONTAINS:

LH	A CODE TO DETERMINE TYPE OF CONSTANT: 1 - XWD 2 - BYTE POINTER 3 - ASCII 4 - SIXBIT 5 - 1-WORD DECIMAL 6 - 2-WORD DECIMAL 7 - FLOATING POINT 1Ø- OCTAL
RH	NUMBER OF WORDS CONTAINING DATA

DATA WORDS ARE DESCRIBED IN MEMO 1ØØ-35Ø-Ø1Ø, ENTITILED "FILE STRUCTURE FOR THE PDP-1Ø COBOL COMPILER", PARAGRAPHS 2.8.4, 2.8.5 AND 2.8.6.

13. TAGTAB - GENERATED TAG TABLE

13.1 USED BY: PHASES E,G
ENTRY SIZE: HALF-WORDS
INITIAL CORE
ALLOCATION: OVERLAYS GRPTAB

13.2 CONTENTS

TAGTAB CONTAINS THE OBJECT TIME LOCATION FOR EACH GENERATED TAG OF THE FORM %NNNNN, IN ORDER BY NUMBER (E.G. %ØØØØØ IN LH OF WORD 1, %ØØØØ1 IN RH OF WORD 1, %ØØØ2Ø IN LH OF WORD 9). FOR EACH HALF-WORD, THE HIGH-ORDER BIT IS Ø IF THE ADDRESS IS RELATIVE TO THE RESIDENT AREA, AND 1 IF RELATIVE TO THE NON-RESIDENT AREA.

14. ALTAB - ALTER TABLE

14.1 USED BY: PHASE E
ENTRY SIZE: 1 WORD
INITIAL CORE
ALLOCATION: 2Ø WORDS

14.2 CONTENTS

ALTAB CONTAINS INFORMATION TO AID IN PROCESSING GO'S WHICH ARE ALTERED.
INFORMATION WILL BE WRITTEN OUT ONTO THE IMPURE ASSEMBLY INPUT (FOR RESIDENT SEGMENT), OR NON-RESIDENT ASSEMBLY INPUT FILES (FOR NON-RESIDENT SEGMENTS) WHEN A SEGMENT BREAK OCCURS.

14.3 DETAILED DESCRIPTION

BIT Ø	RH IS PROTAB LINK (Ø), OR SPECIAL TAG (1).
BITS 1-2Ø	NOT USED
BITS 21-35	RELATIVE ADDRESS OF PROTAB ENTRY, OR SPECIAL TAG NUMBER.

15. SECTAB - SEGMENT TABLE

15.1 USED BY: PHASES D,E,G
ENTRY SIZE: 2 WORDS
INITIAL CORE
ALLOCATION: 2ØØ WORDS

15.2 CONTENTS

SECTAB IS USED FOR TEMPORARY STORAGE DURING PHASE D. DURING PHASES E AND G, SECTAB CONTAINS THE OBJECT TIME STARTING ADDRESSES FOR CERTAIN TABLES. THERE IS ONE ENTRY FOR EACH SEGMENT; THE FIRST ENTRY IS FOR THE RESIDENT SEGMENT, THE REMAINDER FOR EACH NON-RESIDENT SEGMENT.

15.3 DETAILED DESCRIPTION

WORD 1:

	LH	STARTING ADDRESS FOR LITERALS
	RH	NOT USED

WORD 2:

	LH	NUMBER OF ALTAB ENTRIES FOR THIS SEGMENT
	RH	STARTING ADDRESS OF ALTER ADDRESS AT OBJECT TIME.

16. FLOTAB - PROCEDURE DIVISION FLOW TABLE

16.1 USED BY: PHASES D,E
ENTRY SIZE: 2 WORDS
INITIAL CORE
ALLOCATION: 2ØØ WORDS (1ØØ ENTRIES)

16.2 CONTENTS

FLOTAB IS USED TO ENABLE PHASE D TO RESOLVE UNQUALIFIED REFERENCES.

16.3 DETAILED DESCRIPTION

WORD 1:

BIT Ø	PROCEDURE NAME DEFINITION
BIT 1	ENTRANCE PROCEDURE-NAME FOR A PERFORM
BIT 2	EXIT PROCEDURE-NAME FOR A PERFORM
BIT 3	SUBJECT OF AN ALTER
BIT 4	OBJECT OF AN ALTER
BIT 5	OBJECT OF A GO OR GODEP
BITS 6-16	NOT USED
BIT 17	REFERENCE OCCURED IN DECLARATIVES
BITS 18-35	PROTAB LINK

WORD 2:

BIT Ø	NOT USED
BITS 1-15	RELATIVE ADDRESS OF NAMTAB ENTRY
BITS 16-28	LINE NUMBER
BITS 29-35	CHARACTER POSITION

Ø17-Ø4

PDP-1Ø COBOL OPERATING SYSTEM
SUBROUTINE CALLING SEQUENCES

THE INFORMATION IN THIS MEMORANDUM IS SUBJECT TO CHANGE WITHOUT NOTICE AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT CORPORATION.

1. GENERAL

1.1 TYPES OF SUBROUTINES - THE OPERATING SYSTEM SUBROUTINES FALL INTO 6 BROAD CATEGORIES:

1) INPUT-OUTPUT

2) COMPARISON ROUTINES USED BY THE "IF" VERB

3) DOUBLE-PRECISION ARITHMETIC

4) CONVERSION OF DATA FROM ONE USAGE TO ANOTHER

5) EXPONENTIATION

6) MISCELLANEOUS

1.2 CALLING SEQUENCE USED BY COBOL OBJECT CODE - THE MAJORITY OF THE OPERATING SYSTEM SUBROUTINES ARE CALLED WITH UUO'S, THE REMAINDER WITH A PUSHJ 17,. A UUO HANDLER DETERMINES WHICH ROUTINE IS WANTED BY EXAMINING THE OP-CODE AND ACCUMULATOR FIELDS OF THE UUO. THE UUO HANDLER THEN CALLS THE APPROPRIATE SUBROUTINE WITH A PUSHJ 17,. THE UUO MAY BE FOLLOWED BY ONE OR MORE PARAMETERS, DEPENDING UPON THE SUBROUTINE TO BE ENTERED. ALL SUBROUTINES WILL RETURN TO A PLACE IN THE UUO HANDLER WHICH WILL THEN RETURN TO THE MAIN-LINE OBJECT CODE AT THE LOCATION FOLLOWING THE LAST PARAMETER. THE RETURN POINTS IN THE UUO HANDLER ARE:

1) RET.1 - RETURNS TO THE LOCATION OF THE UUO +1

2) RET.2 - RETURNS TO THE LOCATION OF THE UUO +2

3) RET.3 - RETURNS TO THE LOCATION OF THE UUO +3

ONE OF THE FIRST THINGS DONE BY THE UUO HANDLER IS TO PUT THE UUO INTO ACCUMULATOR 16. THE LOCATION OF A BLOCK OF FROM ONE TO THREE PARAMETERS IS SPECIFIED IN THE ADDRESS FIELD OF ACCUMULATOR 16; THAT LOCATION IS USED TO LOAD FROM ONE TO THREE ACCUMULATORS. THE UUO HANDLER HAS THREE ENTRY POINTS TO AID IN LOADING THE ACCUMULATORS:

1) SET.1 - LOAD AC 13

2) SET.2 - LOAD AC'S 13 & 14.

3) SET.3 - LOAD AC'S 13, 14 & 15

(1. GENERAL CONT'D)

1.3 CALLING SEQUENCE FROM NON-COBOL OBJECT CODE.

THE UUO USED BY COBOL IS REPLACED BY THE FOLLOWING CODE:

```
        MOVEI   16,<ADDRESS OF PARAMETERS>
        PUSHJ   17,<ROUTINE>
```

1.4 LOCATION TABLES USED BY UUO HANDLER

THE UUO HANDLER USES NINE TABLES TO CONVERT THE UUO OP-CODE AND ACCUMULATOR FIELDS INTO SUBROUTINE ADDRESSES. THESE TABLES ARE GENERATED AS PART OF THE OBJECT PROGRAM BY THE COMPILER. EACH ENTRY IN THE TABLES IS A HALF-WORD CONTAINING THE ADDRESS OF A SUBROUTINE, IF THAT SUBROUTINE IS NEEDED BY THE OBJECT CODE, OR ZEROES IF THAT SUBROUTINE IS NOT NEEDED. THE UUO HANDLER USES TABLE UUO.1,...., UUO.9 WHEN THE OP-CODE IS ØØ1,....,Ø11 RESPECTIVELY. THE ACCUMULATOR FIELD IS THEN USED TO DETERMINE WHICH ENTRY CONTAINS THE DESIRED ADDRESS. IF THE ACCUMULATOR FIELD CONTAINS ØØ, THE LEFT-HALF OF THE FIRST WORD IS USED; IF THE ACCUMULATOR FIELD CONTAINS 17, THE RIGHT-HALF OF THE EIGHTH WORD IS USED, ETC.

INPUT-OUTPUT ROUTINES

2.1 A DETAILED DESCRIPTION OF THE TABLES USED BY THESE SUBROUTINES MAY BE FOUND IN CHAPTER 8 OF THE COBOL MANUAL.

2.2 OPEN.

2.2.1 USE - OPEN. IS USED TO INITIALIZE A FILE FOR LATER PROCESSING.

2.2.2 CALLING SEQUENCE

```
OP-CODE:          ØØ1
AC-FIELD:         BITØ - OPEN FOR OUTPUT
                  BIT1 - OPEN FOR INPUT
                  BIT2 - REWIND CPTION, 1=NO REWIND
                  BIT3 - ALWAYS Ø
ADDRESS FIELD:    ADDRESS OF AN OBJECT-TIME FILE TABLE
```

2.3 CLOSE.

2.3.1 USE - FINALIZE THE PROCESSING OF A FILE

2.3.2 CALLING SEQUENCE

```
OPCODE: ØØ1
AC-FIELD:         BITØ - CLOSE FILE(Ø) OR REEL(1)
                  BIT1 - "CLOSE WITH LOCK"
                  BIT2 - REWIND OPTION, 1=NO REWIND
                  BIT3 - ALWAYS 1
ADDRESS FIELD:    ADDRESS OF AN OBJECT-TIME FILE TABLE
```

(2. INPUT-OUTPUT ROUTINES CONT'D)

2.4 DSPLY.

2.4.1 USE - DISPLAY A FIELD UPON USER'S CONSOLE

2.4.2 CALLING SEQUENCE

OPCODE: ØØ2
AC-FIELD: ØØ
ADDRESS-FIELD: ADDRESS OF THE PARAMETER

2.4.3 PARAMETER

BITS Ø-5: THE BYTE POINTER RESIDUE FOR THE BYTE PRECEDING THE FIRST CHARACTER OF THE FIELD TO BE DISPLAYED.

BIT 6: THE FIELD IS NUMERIC, SUPPRESS LEADING SPACES

BIT 7: PUT OUT A CARRIAGE-RETURN, LINE-FEED AFTER THE FIELD.

BITS 8-17: SIZE OF THE FIELD.

BITS 18-35: LOCATION CONTAINING THE FIRST CHARACTER OF THE FIELD.

(2. INPUT-OUTPUT ROUTINES CONT'D)

2.5 ACEPT.

2.5.1 USE - READ A FIELD FROM THE USER'S CONSOLE

2.5.2 CALLING SEQUENCE

OP-CODE:	ØØ2
AC-FIELD:	Ø1
ADDRESS-FIELD:	ADDRESS OF THE PARAMETER

2.5.3 PARAMETER FOR NUMERIC FIELDS

BITS Ø-5:	NOT USED
BIT 6:	ALWAYS 1
BIT 7:	SKIP TO END OF LINE AFTER ACCEPTING
BITS 8-17:	SIZE OF FIELD
BITS 18-35:	NUMBER OF DECIMAL PLACES

THE RESULT IS RETURNED IN ACCUMULATORS Ø&1.

2.5.4 PARAMETER FOR NON-NUMERIC FIELDS

BITS Ø-5:	THE BYTE POINTER RESIDUE FOR THE BYTE PRECEDING THE FIRST CHARACTER INTO WHICH TO PLACE THE DATA.
BIT 6:	ALWAYS Ø
BIT 7:	SKIP TO END OF LINE AFTER ACCEPTING
BITS 8-17:	SIZE OF THE FIELD
BITS 18-35:	LOCATION TO CONTAIN THE FIRST CHARACTER OF THE FIELD.

(2. INPUT-OUTPUT ROUTINES CONT'D)

2.6 READ.

2.6.1 USE - READ A RECORD FROM A FILE

2.6.2 CALLING SEQUENCE

WORD 1:
OP-CODE: ØØ2
AC-FIELD: Ø2
ADDRESS-FIELD: ADDRESS OF AN OBJECT-TIME FILE TABLE

WORD 2:
NORMAL RETURN

WORD 3:
READ. RETURNS HERE IF "AT END" OR "INVALID KEY" PATH IS TO BE TAKEN

2.7 WRITE.

2.7.1 USE - WRITE A RECORD ONTO A FILE WITH NO "ADVANCING".

2.7.2 CALLING SEQUENCE

WORD 1:
OP-CODE: ØØ2
AC-FIELD: Ø3
ADDRESS-FIELD: ADDRESS OF AN OBJECT-TIME FILE TABLE

WORD 2: PRESENT ONLY FOR FILES WITH VARIABLE LENGTH RECO
RDS

BITS Ø-11	SIZE OF RECORD, IN CHARACTERS
BITS 12-35	NOT USED

WORD 3:
NORMAL RETURN

WORD 4:
IF "ACCESS MODE" IS RANDOM, RETURN HERE IF "INVALID KEY" PATH IS TO BE TAKEN.

(2. INPUT-OUTPUT ROUTINES CONT'D)

2.8 WADV.

2.8.1 USE - WRITE A RECORD ONTO A FILE, WITH "ADVANCING".

2.8.2 CALLING SEQUENCE

WORD 1:
OP-CODE: ØØ2
AC-FIELD: Ø4
ADDRESS-FIELD: ADDRESS OF AN OBJECT-TIME FILE TABLE

WORD 2:
BITS Ø-11 RECORD SIZE, IN CHARACTERS
BIT 12 RH HAS A CONSTANT (Ø) OR AN ADDRESS (1).

BIT 13 WRITE BEFORE (1) OR AFTER (Ø) ADVANCING

BITS 14-17 THIS IS THE CHANNEL IN A PRINTER
CONTROL-TAPE TO WHICH TO ADVANCE.

BITS 18-35 IF BIT 12 IS Ø, THIS IS THE NUMBER OF CHARACTERS
TO EMIT.
IF BIT 12 IS 1, THIS IS THE ADDRESS OF A LOCATIO
N CONTAINING
THE NUMBER OF CHARACTERS TO EMIT.

2.9 SEEK.

2.9.1 USE - MOVE AN ARM ON A DISK-PACK

2.9.2 CALLING SEQUENCE
OP-CODE: ØØ2
AC-FIELD: Ø5
ADDRESS-FIELD: ADDRESS OF AN OBJECT-TIME FILE TABLE

3. COMPARISON ROUTINES - OP-CODE ØØ3

3.1 COMP.

3.1.1 USE - COMPARE TWO FIELDS FOR RELATIVE VALUE; ASCII VS. ASCII OR SIXBIT VS. SIXBIT

3.1.2 CALLING SEQUENCE

WORD 1:

AC-FIELD:	ØØ
ADDRESS-FIELD:	ADDRESS OF FIRST PARAMETER

WORD 2: RETURN IF "A"<"B"

WORD 3: RETURN IF "A">"B"

WORD 4: RETURN IF "A"="B"

3.1.3 PARAMETERS

WORD 1: BYTE POINTER FOR OPERAND "A"

WORD 2:

BITS Ø-5	BYTE POINTER RESIDUE FOR OPERAND "B"
BITS 6-17	SIZE OF BOTH "A" AND "B"
BITS 18-35	ADDRESS OF LOCATION CONTAINING FIRST CHARACTER OF "B"

3.2 CMP.76

3.2.1 USE - COMPARE AN ASCII FIELD VS. A SIXBIT FIELD

3.2.2 CALLING SEQUENCE - IDENTICAL TO 3.1.2 EXCEPT THAT AC-FIELD IS Ø1

3.2.3 PARAMETERS - IDENTICAL TO 3.1.3

(3. COMPARISON ROUTINES CONT'D)

3.3 SPAC.6

3.3.1 USE - COMPARE A SIXBIT FIELD AGAINST SPACES

3.3.2 CALLING SEQUENCES

WORD 1:
AC-FIELD: Ø2
ADDRESS-FIELD: ADDRESS OF A PARAMETER

WORD 2: RETURN IF THE FIELD IS NOT ALL SPACES

WORD 3: RETURN IF THE FIELD IS ALL SPACES

3.3.3 PARAMETER

BITS Ø-5	BYTE POINTER RESIDUE FOR THE FIELD
BIT 6	FIELD IS SIGNED
BITS 7-17	SIZE OF THE FIELD
BITS 18-35	ADDRESS OF THE LOCATION CONTAINING THE FIRST CHARACTER OF THE FIELD

3.4 NUM.6

3.4.1 USE - DETERMINE IF A SIXBIT FIELD IS NUMERIC, I.E. CONTAINS ONLY THE DIGITS Ø-9.

3.4.2 CALLING SEQUENCE

WORD 1:
AC-FIELD: Ø3
ADDRESS-FIELD: ADDRESS OF A PARAMETER

WORD 2: RETURN IF THE FIELD IS NOT NUMERIC

WORD 3: RETURN IF THE FIELD IS NUMERIC

3.4.3 PARAMETER - SEE 3.3.3

(3. COMPARISON ROUTINES CONT'D)

3.5 ALF.6

3.5.1 USE - DETERMINES IF A SIXBIT FIELD IS ALPHABETIC, I.E. CONTAINS ONLY THE LETTERS A-Z AND BLANK.

3.5.2 CALLING SEQUENCE

```
WORD 1:
          AC-FIELD:        Ø4
          ADDRESS-FIELD:   ADDRESS OF A PARAMETER

WORD 2: RETURN IF THE FIELD IS NOT ALPHABETIC

WORD 3: RETURN IF THE FIELD IS ALPHABETIC
```

3.5.3 PARAMETER - SEE 3.3.3

3.6 ZERO.6

3.6.1 USE - DETERMINE IF A SIXBIT FIELD CONTAINS THE VALUE ZERO.

3.6.2 CALLING SEQUENCE

```
WORD 1:
          AC-FIELD:        Ø5
          ADDRESS-FIELD:   ADDRESS OF A PARAMETER

WORD 2: RETURN IF THE FIELD IS NOT ZERO

WORD 3: RETURN IF THE FIELD IS ZERO
```

3.6.3 PARAMETER - SEE 3.3.3

(3. COMPARISON ROUTINES CONT'D)

3.7 POS.6

3.7.1 USE - DETERMINE IF A SIXBIT FIELD CONTAINS A POSITIVE VALUE

3.7.2 CALLING SEQUENCE

```
WORD 1:
        AC-FIELD:       Ø6
        ADDRESS-FIELD:  ADDRESS OF A PARAMETER

WORD 2: RETURN IF THE FIELD IS NOT POSITIVE

WORD 3: RETURN IF THE FIELD IS POSITIVE
```

3.7.3 PARAMETER - SEE 3.3.3

3.8 NEG.6

3.3.1 USE - DETERMINE IF A SIXBIT FIELD CONTAINS A NEGATIVE VALUE

3.8.2 CALLING SEQUENCE

```
WORD 1:
        AC-FIELD:       Ø7
        ADDRESS-FIELD:  ADDRESS OF A PARAMETER

WORD 2: RETURN IF THE FIELD IS NOT NEGATIVE

WORD 3: RETURN IF THE FIELD IS NEGATIVE
```

3.8.3 PARAMETER - SEE 3.3.3

(3. COMPARISON ROUTINES CONT'D)

3.9 SPAC.7, NUM.7, ALF.7, ZERO.7, POS.7, NEG.7 ARE IDENTICAL TO SPAC.6,...,NEG.6 EXCEPT THAT THE FIELD IS ASCII. THE AC-FIELD FOR THE UUO IS 1Ø THRU 15, RESPECTIVELY.

3.1Ø COMP.D

3.1Ø.1 USE - COMPARE TWO DOUBLE-PRECISION FIELDS FOR RELATIVE VALUE

3.1Ø.2 CALLING SEQUENCE - "A" OPERAND IS IN ACCUMULATORS Ø AND 1

```
WORD 1:
        AC-FIELD:        16
        ADDRESS-FIELD:   ADDRESS OF THE FIRST WORD OF
                         A TWO-WORD "B" OPERAND

WORD 2: RETURN IF "A"<"B"

WORD 3: RETURN IF "A">"B"

WORD 4: RETURN IF "A"="B"
```

DOUBLE-PRECISION ARITHMETIC

4.1 NOMENCLATURE

THE DOUBLE-PRECISION ROUTINES ARE DIVIDED INTO FIVE CATEGORIES:

1) ADD. ADD

2) SUB. SUBTRACT

3) MUL. MULTIPLY

4) DIV. DIVIDE

5) NEG.,MAG. NEGATE, ABSOLUTE VALUE

THE FIRST 4 CATEGORIES HAVE TWO DIGITS APPENDED TO DENOTE THE SIZE OF THE OPERANDS. THE FIRST DIGIT IS THE SIZE OF THE OPERAND CONTAINED IN THE ACCUMULATORS, THE SECOND DIGIT IS THE SIZE OF THE OPERAND IN MEMORY. FOR EXAMPLE, ADD.21 ADDS A SINGLE-PRECISION NUMBER TO ACCUMULATORS AC & AC+1. NEG. AND MAG. ALWAYS WORK WITH DOUBLE-PRECISION NUMBERS; THE OPERAND IS IN MEMORY; THE RESULT PUT INTO THE AC'S.

4.2 CALLING SEQUENCE

OP-CODE:	SEE BELOW
AC-FIELD:	AC OF ONE OPERAND, AND OF RESULT
ADDRESS-FIELD:	ADDRESS OF THE OPERAND

NAME	OP-CODE	FUNCTION
NEG.	Ø21	NEGATE A DOUBLE PRECISION WORD
MAG.	Ø22	GET MAGNITUDE OF DOUBLE-PRECSION NUMBER
ADD.12	Ø23	ADD DOUBLE TO SINGLE
ADD.21	Ø24	ADD SINGLE TO DOUBLE
ADD.22	Ø25	ADD DOUBLE TO DOUBLE
SUB.12	Ø26	SUBTRACT DOUBLE FROM SINGLE
SUB.21	Ø27	SUBTRACT SINGLE FROM DOUBLE
SUB.22	Ø3Ø	SUBTRACT DOUBLE FROM DOUBLE
MUL.12	Ø31	MULTIPLY SINGLE BY DOUBLE
MUL.21	Ø32	MULTIPLY DOUBLE BY SINGLE
MUL.22	Ø33	MULTIPLY DOUBLE BY DOUBLE
DIV.11	Ø34	DIVIDE SINGLE BY SINGLE
DIV.12	Ø35	DIVIDE SINGLE BY DOUBLE
DIV.21	Ø36	DIVIDE DOUBLE BY SINGLE
DIV.22	Ø37	DIVIDE DOUBLE BY DOUBLE

5. CONVERSION ROUTINES

5.1 NOMENCLATURE

NO GENERAL NAMING SCHEME

5.2 PARAMETERS - ROUTINES CONVERTING TO OR FROM DISPLAY FIELDS USE PARAMETERS DESCRIBING THE FIELDS

5.2.1 PARAMETER A - 2 WORDS

WORD 1: BYTE POINTER FOR "FROM" FIELD
WORD 2:

BITS Ø-5	BYTE POINTER RESIDUE FOR "TO" FIELD
BIT 6	ALWAYS Ø
BITS 7-17	SIZE OF BOTH FIELDS
BITS 18-35	ADDRESS OF LOCATION CONTAINING THE FIRST CHARACTER OF THE "TO" FIELD

5.2.2 PARAMETER B - 1 WORD

BITS Ø-5	BYTE POINTER RESIDUE FOR "FROM" FIELD
BIT 6	A 1 IF THE FIELD HAS AN OPERATIONAL SIGN
BITS 7-12	NOT USED
BITS 13-17	SIZE OF "FROM" FIELD
BITS 18-35	ADDRESS OF LOCATION CONTAINING THE FIRST CHARACTER OF THE "FROM" FIELD

5.2.4 PARAMETER C - 2 WORDS

WORD 1: BYTE POINTER TO SIGN BYTE OF 'FROM' FIELD
WORD 2: BYTE POINTER TO SIGN BYTE OF 'TO' FIELD

(5. CONVERSION ROUTINES CONT'D)

5.3 FUNCTION

C.D6D7	CONVERT SIXBIT TO ASCII
C.D7D6	CONVERT ASCII TO SIXBIT
FIX.	CONVERT FLOATING-POINT TO 2-WORD COMPUTATIONAL
FLOT.1	CONVERT 1-WORD COMPUTATIONAL TO FLOATING-POINT
FLOT.2	CONVERT 2-WORD COMPUTATIONAL TO FLOATING-POINT
GD6.	CONVERT SIXBIT TO COMPUTATIONAL
GD7.	CONVERT ASCII TO COMPUTATIONAL
PD6.	CONVERT COMPUTATIONAL TO SIXBIT
PD7.	CONVERT COMPUTATIONAL TO ASCII
SIGN.	MOVE SIGN FROM ONE DISPLAY FIELD TO ANOTHER
MOVE.	MOVE ASCII - ASCII, OR SIXBIT - SIXBIT

5.4 CALLING SEQUENCES

ROUTINE	OP-CODE	AC-FIELD	ADDRESS-FIELD	FROM	TO
MOVE.	ØØ4	ØØ	PARAMETER A	MEMORY	MEMORY
C.D6D7	ØØ4	Ø1	PARAMETER A	MEMORY	MEMORY
C.D7D6	ØØ4	Ø2	PARAMETER A	MEMORY	MEMORY
SIGN.	ØØ4	Ø3	PARAMETER C	MEMORY	MEMORY
FIX.	Ø1Ø		OPERAND	MEMORY	AC
FLOT.1	Ø13		OPERAND	MEMORY	AC
FLOT.2	Ø14		OPERAND	MEMORY	AC
PD6.	Ø15		OPERAND	AC	MEMORY
PD7.	Ø16		OPERAND	AC	MEMORY
GD6.	Ø17		OPERAND	MEMORY	AC
GD7.	Ø2Ø		OPERAND	MEMORY	AC

6. MISCELLANEOUS UUO'S

6.1 OVLAY.

6.1.1 USE - READ IN AN OVERLAY SEGMENT, RESET THE OBJECT OF ALL ALTERED GO'S (IF REQUIRED), AND TRANSFER CONTROL TO A LOCATION WITHIN THE OVERLAY SEGMENT.

6.1.2 CALLING SEQUENCE

```
                  OP-CODE:          ØØ5
                  AC-FIELD:         12
                  ADDRESS-FIELD:    ADDRESS OF THE PARAMETER
```

6.1.3 PARAMETER

```
    BITS Ø-17       ADDRESS OF LOCATION TO WHICH
                    CONTROL IS TO BE TRANSFERRED

    BITS 18-19      NOT USED
    BITS 2Ø-26      SEGMENT PRIORITY NUMBER FOR THE SEGMENT
                    CONTAINING THE GO
    BITS 27-28      NOT USED
    BITS 29-35      SEGMENT PRIORITY NUMBER FOR THE
                    OVERLAY SEGMENT
```

6.2 PERF.

6.2.1 USE - SET UP A PERFORM

6.2.2 CALLING SEQUENCE

```
                  OP-CODE:          Ø12
                  AC-FIELD:         ØØ - ANY AND ALL SUBSEQUENT OVERLAYS ALL
OWED
                                    Ø1 - NO OVERLAYS ALLOWED
                                    17 - ALL OVERLAYS ALLOWED (SPECIAL FOR
                                         DECLARATIVES)

                  ADDRESS-FIELD:    ADDRESS OF THE WORD TO CONTAIN RETURN AD
DRESS
```

6.3 EXIT.

6.3.1 USE - RETURN FROM PERFORMED CODE

6.3.2 CALLING SEQUENCE

```
                  OP-CODE:          ØØ5
                  AC-FIELD:         12
                  ADDRESS-FIELD:    LOCATION OF A WORD CONTAINING RETURN ADD
RESS
```

(6. MISCELLANEOUS UUO'S CONT'D)

6.4 EXAM.

6.4.1 USE - PERFORM THOSE OPERATIONS REQUIRED BY THE COBOL "EXAMINE" VERB.

6.4.2 CALLING SEQUENCE

WORD 1:

OP-CODE:	ØØ5
AC-FIELD:	Ø2
ADDRESS-FIELD:	ADDRESS OF A LOCATION CONTAINING A BYTE POINTER TO THE FIELD TO BE EXAMINED

WORD 2:

BIT 1	1=SIGNED NUMERIC
BITS 2-5	NOT USED
BITS 6-17	SIZE OF THE FIELD
BIT 18	EXAMINE FOR "LEADING"
BIT 19	EXAMINE FOR "FIRST"
BIT 2Ø	EXAMINE FOR "UNTIL FIRST"
BIT 21	"REPLACING"
BITS 22-28	THE CHARACTER TO EXAMINE FOR. THIS IS A SIXBIT CHARACTER IF THE FIELD IS IN DISPLAY-6 USAGE, AN ASCII CHARACTER IF THE FIELD IS IN DISPLAY-7 USAGE.
BITS 29-35	THE CHARACTER TO REPLACE WITH, AGAIN EITHER IN SIXBIT OR ASCII.

6.4.3 TALLY

THE RESULT OF TALLYING IS RETURNED IN ACCUMULATOR Ø.

(6. MISCELLANEOUS UUO'S CONT'D)

6.5 EDIT.S, EDIT.U

6.5.1 USE - MOVE A DISPLAY FIELD FROM ONE LOCATION TO ANOTHER, PERFORMING EDITING AS DIRECTED BY THE COBOL PICTURE CLAUSE. EDIT.S IS USED IF BOTH THE "FROM" FIELD AND THE "TO" FIELD ARE SIGNED; EDIT.U IS USED OTHERWISE.

6.5.2 CALLING SEQUENCE

WORD 1:		
	OP-CODE:	ØØ5
	AC-FIELD:	ØØ FOR EDIT.S, Ø1 FOR EDIT.U
	ADDRESS-FIELD:	ADDRESS OF THE FIRST OF TWO OR THREE PARAMETERS
WORD 2:		
	BITS Ø-5	BYTE POINTER RESIDUE FOR THE PICTURE MASK
	BITS 6-11	THE PICTURE SIGN CHARACTER IN SIXBIT. IF THE SIGN IS "-", THIS FIELD IS BLANK.
	BITS 12-17	THE PICTURE SUPPRESSION OR FLOATING CHARACTER, IN SIXBIT. IF THE CHARACTER IS "Z" OR "-", THIS FIELD IS BLANK.
	BITS 18-35	THE ADDRESS OF THE LOCATION CONTAINING THE FIRST MASK CHARACTER.

6.5.3 PARAMETERS

EDIT.S HAS ALL THREE PARAMETERS, EDIT.U HAS ONLY THE SECOND AND THIRD.

PARAMETER ONE - BYTE POINTER TO THE SIGN CHARACTER OF THE "FROM" FIELD

PARAMETER TWO - BYTE POINTER TO THE CHARACTER PRECEDING THE FIRST CHARACTER OF THE "FROM" FIELD

PARAMETER THREE - BYTE POINTER TO THE CHARACTER PRECEDING THE FIRST CHARACTER OF THE "TO" FIELD. IF THE RESULT FIELD IS "BLANKED WHEN ZERO", BIT 12 WILL BE A 1.

(6. MISCELLANEOUS UUO'S CONT'D)

6.5.4 MASK FIELD

THE MASK IS A STRING OF 4-BIT BYTES WHICH DIRECT THE EDITING. THE VALUE OF THOSE BYTES ARE:

ØØ	INSERT AN ALPHA CHARACTER (X,A)
Ø1	INSERT A NUMERIC CHARACTER (9)
Ø2	ZERO SUPPRESS (Z,*)
Ø3	FLOAT ($$,++,--)
Ø4	INSERT A COMMA (,)
Ø5	INSERT A BLANK (B)
Ø6	INSERT A ZERO (Ø)
Ø7	INSERT A CURRENCY SIGN ($)
1Ø	INSERT A SIGN (+,-)
11	INSERT A DECIMAL POINT (.)
12	INSERT A CREDIT SYMBOL (CR)
13	INSERT A DEBIT SYMBOL (DB)
14-16	UNUSED
17	TERMINATE EDITING

(6. MISCELLANEOUS UUO'S CONT'D)

6.6 SUBSC.

6.6.1 USE - CREATE A BYTE POINTER TO AN ELEMENT IN AN ARRAY
AND PLACE IT IN ACCUMULATOR 12

6.6.2 CALLING SEQUENCE

OP-CODE:	ØØ5
AC-FIELD:	Ø3
ADDRESS-FIELD:	ADDRESS OF PARAMETER BLOCK

6.6.3 PARAMETERS

WORD 1: BYTE POINTER FOR ARRAY ELEMENT (1,1,1)

WORD 2:

BITS Ø-17	NUMBER OF SUBSCRIPTS
BITS 18-23	NOT USED
BITS 24-35	A CONSTANT TO BE PLACED IN BITS 6-17 OF RESULTIN

G

BYTE POINTER.

WORDS 3-N: TWO WORDS FOR EACH SUBSCRIPT

1)	LH	ADDRESS OF "DEPENDING" VARIABLE (Ø IF NONE)
	RH	ADDRESS OF THE SUBSCRIPT
2)	LH	SIZE OF THE ITEM HAVING THE OCCURS CLAUSE
	RH	BIT 18: SUBSCRIPT IS LITERAL (Ø) OR DATA-NAME (1

)

BITS 19-2Ø	NOT USED
BITS 21-35	MAXIMUM ALLOWED VALUE FOR SUBSCRIPT ("OCCURS" AMOUNT)

6.7 SIZE.1, SIZE.2, SIZE.3

6.7.1 USE - DETERMINE IF AN ITEM WILL CAUSE A SIZE ERROR.
SIZE.1 IS USED IF AC'S ARE 1 WORD, LITERAL IS 1 WORD.
SIZE.2 IS USED IF AC'S ARE 2 WORDS, LITERAL IS 1 WORD.
SIZE.3 IS USED IF AC'S ARE 2 WORDS, LITERAL IS 2 WORDS.

6.7.2 CALLING SEQUENCE

OP-CODE:	ØØ5
AC-FIELD:	Ø4,Ø5,Ø6 RESPECTIVELY
ADDRESS-FIELD:	ADDRESS OF ACCUMULATOR (THE HI-ORDER ONE IN THE

CASE OF 2).

6.7.3 PARAMETER WORD (FOLLOWING UUO)

LH	ADDRESS OF RETURN LOCATION IF SIZE ERROR OCCURS
RH	ADDRESS OF LITERAL TO BE COMPARED AGAINST

(6. MISCELLANEOUS UUO'S CONT'D)

6.8 E.C3C1,E.C3C2

6.8.1 USE - EXPONENTIATE A FLOATING-POINT NUMBER. NOTE THAT THE RESULT IS ALWAYS FLOATING POINT.

E.C3C1 RAISES A FLOATING-POINT NUMBER TO AN INTEGRAL POWER.
E.C3C2 RAISES A FLOATING-POINT NUMBER TO A FLOATING-POINT POWER.

6.8.2 CALLING SEQUENCE:

OP-CODE:	ØØ5
AC-FIELD:	Ø7 FOR E.C3C1, 1Ø FOR E.C3C2
ADDRESS-FIELD:	ADDRESS OF POWER

6.9 ULOSE.

6.9.1 USE - CALLED WHEN COMPILER GENERATES CODE WHICH REFERENCES A NON-EXISTENT ROUTINE.

6.9.2 CALLING SEQUENCE:

OP-CODE:	ANY NOT USED BY OTHER ROUTINES
AC-FIELD:	ANY NOT USED BY OTHER ROUTINES WITH SAME OP-CODE
ADDRESS-FIELD:	NOT USED

7. ROUTINES NOT CALLED VIA UUO'S

7.1 STOPR.

7.1.1 USE - CALLED WHEN "STOP RUN" EXECUTED. ALL OPEN FILES ARE CLOSED, AND CONTROL IS TRANSFERED TO THE MONITOR WITH A CALL [SIXBIT "EXIT"] UUO.

7.1.2 CALLING SEQUENCE - PUSHJ 17,STOPR.

7.2 STOP.

7.2.1 USE - CALLED WHEN "STOP LITERAL" EXECUTED. THE ROUTINE WAITS FOR THE OPERATOR TO TYPE "CONTINUE", THEN RETURNS TO THE CALLING ROUTINE.

7.2.2 CALLING SEQUENCE - PUSHJ 17,STOP.

7.3 KILL.

7.3.1 USE - TERMINATE THE EXECUTION OF THE PROGRAM BECAUSE OF SOME ERROR.

7.3.4 CALLING SEQUENCE

AN ERROR MESSAGE IS TYPED, USING THE DSPLY. ROUTINE, THEN KILL. IS CALLED WITH PUSHJ 17,KILL.

7.4 GOTO.

7.4.1 USE - PROVIDE AN ERROR EXIT FOR "GO TO" STATEMENTS WHICH DID NOT PROVIDE AN OBJECT PARAGRAPH NAME, AND WERE NOT ALTERED.

7.4.2 CALLING SEQUENCE - PUSHJ 17,GOTO.

7.5 RESET.

7.5.1 USE - INITIALIZE A PROGRAM. SET UP THE PUSH-DOWN POINTER, AND RESET ALL FILES TO THEIR INITIAL STATE. ALLOCATE BUFFER SPACE FOR ALL FILES WHICH SHARE THEIR BUFFER AREAS.

7.5.2 CALLING SEQUENCE - JSP 16,RESET.

(7. MISCELLANEOUS CONT'D).

7.6 KDECL.

7.6.1 USE - USED WHEN USER TRIES TO GO BEYOND DECLARATIVES AT RUN-TIME

7.6.2 CALLING SEQUENCE - PUSHJ 17,KDECL.

7.7 KPROG.

7.7.1 USE - USED WHEN USER TRIES TO GO BEYOND END OF PROGRAM.

7.7.2 CALLING SEQUENCE - PUSHJ 17,KPROG.

8. SORT ROUTINES

8.1 PSORT.

8.1.1 USE - INITIALIZE SORT

8.1.2 CALLING SEQUENCE:

```
                PUSHJ 17,PSORT.
                XWD <NUMBER OF WORDS TO CONTAIN KEYS>,
                                <ADDRESS OF FILE-TABLE FOR SORT FILE>
                XWD <LOCATION OF KEYS>,<LOCATION OF KEY-ASSEMBLY ROUTINE
>
```

8.2 MERGE.

8.2.1 USE - MERGE SORT SCRATCH FILES

8.2.2 CALLING SEQUENCE - PUSHJ 17,MERGE.

8.3 RELES.

8.3.1 USE - RELEASE A RECORD TO PRESORT

8.3.2 CALLING SEQUENCE:

```
                MOVEI 16,<SIZE OF RECORD IN WORDS>
                PUSHJ 17,RELES.
```

8.4 RETRN.

8.4.1 USE - GET A RECORD FROM FINAL MERGE PHASE OF SORT

8.4.2 CALLING SEQUENCE:

```
                PUSHJ 17,RETRN.
                EXIT HERE IF NOT "AT END"
                EXIT HERE IF "AT END"
```

8.5 ENDS.

8.5.1 USE - FINISH UP SORT

8.5.2 CALLING SEQUENCE - PUSHJ 17,ENDS.

8.6 KEY.

8.6.1 USE - ADJUST AN ALPHANUMERIC KEY TO CLEAR SIGN BIT

8.6.2 CALLING SEQUENCE:

```
                PUSHJ 17,KEY.
                EXP <BYTE POINTER TO DISPLAY KEY>
                XWD <SIZE OF FIELD>,<FIRST LOCATION FOR OUTPUT>
```

IF FIELD IS TO BE SORTED IN DESCENDING ORDER, THE SIGN BIT OF THE SECOND PARAMETER IS SET TO 1.

001-02

PROGRAMMING PROJECT SPECIFICATION

SOURCE LIBRARY MAINTENANCE ROUTINE

THE INFORMATION IN THIS MEMORANDUM IS SUBJECT TO CHANGE WITHOUT NOTICE AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT CORPORATION.

Ø.1 OVERALL DESCRIPTION

THE COBOL SOURCE LIBRARY MAINTENANCE ROUTINE WILL MAINTAIN A FILE, ON EITHER DECTAPE OR DISK, OF COBOL SOURCE LANGUAGE USED IN CONJUNCTION WITH THE COBOL COPY VERB.

THE ROUTINE WILL BE CAPABLE OF ADDING, REPLACING AND/OR DELETING SOURCE LANGUAGE DATA ON A FILE, AND LISTING AN ENTRY OF THE FILE.

1. GENERAL SPECIFICATION

1.1 MACHINE REQUIREMENTS

THE MAINTENANCE ROUTINE REQUIRES A PDP-6/1Ø CENTRAL PROCESSOR, A DISKFILE AND A CONSOLE TELETYPE.

1.2 MACHINE OPTIONS

THE ORIGINAL AND/OR UPDATED VERSION OF THE FILE MAY BE PUT ON DECTAPE, WITH A SMALL SACRIFICE OF RUNNING TIME. ANY DEVICE CAPABLE OF HANDLING ASCII OUTPUT MAY BE USED AS THE LISTING DEVICE.

1.3 SYSTEM REQUIRMENTS

A MONITOR WITH DISK CAPABILITIES IS REQUIRED.

1.4 RESIDENT PROGRAMS

THE ROUTINES WILL BE SELF-CONTAINED.

2. DESIGN GOALS

1) THE ROUTINE WILL RUN IN 2K OF USER CORE.

2) THE ROUTINE WILL BE DESIGNED AROUND THE DISK, BUT OTHER DEVICES MAY BE USED WHERE APPROPRIATE.

3) THE ROUTINE WILL RUN UNDER THE CONTROL OF BATCH.

4) THE COMMANDS MAY BE TAKEN FROM ANY DEVICE.

.2 INPUT

2.2.1 INPUT FORMAT

THE INPUT FILE IS A COLLECTION OF COBOL SOURCE LANGUAGE ROUTINES, EACH IDENTIFIED BY A UNIQUE 8-CHARACTER LIBRARY-NAME. THE LIBRARY FILE MUST BE ON A DIRECTORY DEVICE.

THE DATA CONTAINED IN THE LIBRARY IS DIVIDED INTO THREE SECTIONS:

1. THE SOURCE LANGUAGE, IN ASCII. THIS IS A COLLECTION OF NAMED ROUTINES WRITTEN IN COBOL, TO BE REFERENCED BY THE COBOL COPY VERB. THE ROUTINES ARE IN ALPHABETIC ORDER BY LIBRARY-NAME. THE MAY BE AS MANY AS 3869 LIBRARY ROUTINES.

2. A TABLE OF LIBRARY-NAMES, WITH POINTER TO THE DATA IN THE SOURCE LANGUAGE SECTION. THIS TABLE IS CALLED THE FINE TABLE, AND MAY EXTEND OVER AS MANY AS 63 BLOCKS.

3. A ROUGH TABLE POINTING TO THE BLOCKS IN THE FINE TABLE. THIS TABLE IS ONE BLOCK (126 WORDS) LONG.

.2.2 CHARACTER SET

THE COBOL CHARACTER SET IS USED.

2.3 OUTPUT

2.3.1 OUTPUT FORMAT

THE FORMAT OF THE OUTPUT FILES IS IDENTICAL TO THAT FOR THE INPUT FILE (2.2.1).

2.3.2 CHARACTER SET

THE COBOL CHARACTER SET IS USED.

2.4 ORGANIZATION

2.4.1 OPERATIONAL ORGANIZATION

THE MAINTENANCE ROUTINE WILL COPY THE INPUT FILE TO DISK, UPDATING AS IT GOES. UPON COMPLETION OF THE UPDATE, THE NEW LIBRARY WILL BE COPIED TO THE OUTPUT FILE.

2.4.2 INTERNAL ORGANIZATION

ALL SUBROUTINES ARE RESIDENT. THE ROUTINE WILL USE THE CORE UUO TO INCREASE ITS CORE USAGE, IF NECESSARY.

3.1. LOADING PROCEDURE

ONLY ONE OBJECT FILE IS TO BE LOADED, IN ADDITION TO JOBDAT. THE LINKING LOADER IS USED.

3.1.1 CONDITIONAL LOAD - NOT APPLICABLE

3.2 SWITCH SETTINGS - NO CONSOLE SWITCHES ARE USED

3.3 START-UP PROCEDURE

THIS ROUTINE WILL BE ONE OF THE CUSP FILES, AND WILL BE STARTED WITH THE R, RUN OR GET AND START COMMANDS. THERE WILL BE NO REENTER PROCEDURE.

THE OPERATOR WILL SPECIFY THE DEVICES TO BE USED BY TYPING

FILE1,FILE2←FILE3

WHERE FILEN IS OF THE FORM "DEV:NAME.EXT[PROJ,PROG]." FILE1 IS THE FILE TO CONTAIN THE OUTPUT, FILE2 IS THE LISTING FILE AND FILE3 CONTAINS THE LIBRARY TO BE UPDATED. FILE2 NEED NOT BE SPECIFIED, IN WHICH CASE NO LISTING OF CORRECTED ROUTINES WILL BE PRODUCED.

IF FILE-NAME EXTENSIONS ARE NOT SPECIFIED, LIB WILL BE USED FOR FILE1 AND FILE3, LST FOR FILE2.

IF DEVICES ARE NOT SPECIFIED, DSK WILL BE USED.

IF FILENAMES ARE NOT SPECIFIED, LIBARY WILL BE USED.

IF THE INPUT FILE AND OUTPUT FILE HAVE THE SAME FILE-NAME AND EXTENSION, AND ARE BOTH ON THE SAME DEVICE, THE EXTENSION OF THE INPUT FILE WILL BE CHANGED TO BAK AT THE COMPLETION OF THE RUN.

THE FOLLOWING SWITCHES ARE RECOGNIZED:

Z - CLEAR AN OUTPUT DIRECTORY (DECTAPE ONLY).

W - REWIND (LISTING ON MTA ONLY).

3.4 COMMAND LANGUAGE

SIX COMMANDS ARE USED TO POSITION THE INPUT AND SCRATCH FILES:

INSERT LIBRARY-NAME - THE INPUT FILE WILL BE COPIED TO THE SCRATCH FILE, STARTING AT THEIR CURRENT POSITIONS, UNTIL A SOURCE ROUTINE WITH A NAME GREATER THAN THAT SPECIFIED IS ENCOUNTERED. THE NEW NAME WILL BE INSERTEV IN THE FINE TABLE, AND THE PROGRAM WILL AWAIT ANOTHER COMMAND.

DELETE LIBRARY-NAME - THE INPUT FILE WILL BE COPIED TO THE SCRATCH FILE UNTIL A SOURCE ROUTINE OF THE SPECIFIED NAME IS ENCOUNTERED. THE INPUT FILE WILL THEN BE POSITIONED AFTER THAT SOURCE ROUTINE.

REPLACE LIBRARY-NAME - THE PROGRAM WILL DO A DELETE FOLLOWED BY AN INSERT.

CORRECT LIBRARY-NAME - THE INPUT FILE WILL BE COPIED TO THE SCRATCH FILE UNTIL A SOURCE ROUTINE OF THE SPECIFIED NAME IS ENCOUNTERED.

END - THE REMAINDER OF THE INPUT FILE WILL BE COPIED TO THE SCRATCH FILE, THE OUTPUT FILE CREATED, AND THE PROGRAM WILL TERMINATE.

RESTART - THE REMAINDER OF THE INPUT FILE WILL BE COPIED TO THE SCRATCH FILE. THE SCRATCH FILE WILL THEN BECOME THE INPUT FILE, AND A NEW SCRATCH FILE STARTED. THIS ALLOWS THE USER TO UPDATE ROUTINES OUT OF LIBRARY-NAME ORDER.

TYPING /N AFTER THE CORRECT COMMAND WILL CAUSE NEW LINE NUMBERS TO BE APPLIED TO THE OUTPUT VERSION OF THE SOURCE LANGUAGE ROUTINE.

THREE COMMANDS ARE USED TO ALTER THE CONTENTS OF A SOURCE FILE:

DNNNNNN THE INPUT FILE IS COPIED TO THE SCRATCH UNTIL THE SPECIFIED LINE NNNNNN IS ENCOUNTERED. THAT LINE WILL THEN BE SKIPPED.

INNNNNN COBOL - STATEMENT
THE INPUT FILE IS COPIED UNTIL A LINE HAVING A LARGER LINE NUMBER IS ENCOUNTERED, OR UNTIL A NEW SOURCE LANGUAGE ROUTINE IS ENCOUNTERED.
THE COBOL - STATEMENT WILL BE INSERTED AT THAT POINT.

RNNNNNN COBOL - STATEMENT
THE INPUT FILE IS COPIED UNTIL THE SPECIFIED LINE IS ENCOUNTERED. THAT COBOL STATEMENT IS REPLACED BY THE STATEMENT IN THE COMMAND.

3.4.1 EXAMPLES

A LIBRARY EXISTS ON THE DISK CONTAINING THE ROUTINES PAYCOMP, FIND-MP AND MP-DESCR. THE USER WISHES TO DELETE PAYCOMP, CORRECT MP-DESCR AND INSERT A NEW ROUTINE TO BE CALLED JOB-DESC. THE MP-DESCR ROUTINE CONTAINS THE FOLLOWING SOURCE STATEMENTS:

```
     ØØØØ1Ø              LABEL RECORDS ARE OMITTED
     ØØØØ2Ø              DATA RECORD IS MP-RECORD.
```

THE DIALOG AT THE CONSOLE MIGHT BE:

```
     R LIBARY
     LIBARY.NEW←LIBARY.OLD
     INSERT              JOB-DESC
     IØØØØ1Ø             LABEL RECORDS ARE STANDARD;
     IØØØØ2Ø             VALUE OF ID IS "JOBS";
     IØØØØ3Ø             DATA RECORD IS JOB-RECORD.
     CORRECT             MP-DESCR/N
     IØØØØØ5             BLOCK CONTAINS 5 RECORDS
     DELETE              PAYCOMP
     END
```

FILE LIBARY.NEW WILL NOW CONTAIN:
1) ROUTINE FIND-MP
2) ROUTINE JOB-DESC
3) ROUTINE MP-DESCR, ALTERED TO APPEAR AS

```
     ØØØØ1Ø              BLOCK CONTAINS 5 RECORDS
     ØØØØ2Ø              LABEL RECORDS ARE OMITTED
     ØØØØ3Ø              DATA RECORD IS MP-RECORD.
```

3.5 OPERATION - SEE 3.3

3.6 ERROR RECOVERY

3.6.1 INPUT ERRORS

IF THE INPUT FILE IS NOT A LIBRARY FILE, THE PROGRAM WILL TYPE:

```
? INCORRECT LIBRARY FILE FORMAT
```

AND TERMINATE WITH A CALL [SIXBIT /EXIT/].

THE INPUT FILE IS NOT A LIBRARY FILE IF ONE OF THE FOLLOWING CONDITIONS EXIST:

1) THE ROUGH TABLE IS NOT IN ORDER BY LIBRARY-NAME.
2) A FINE TABLE IS NOT IN ORDER BY LIBRARY-NAME.
3) A LIBRARY ROUTINE IS NOT IN ORDER BY LINE NUMBER.

3.6.2 OPERATOR ERRORS

IF AN IMPROPER COMMAND IS DETECTED, AN ERROR MESSAGE WILL BE TYPED, AND THE PROGRAM WILL LOOK FOR ANOTHER COMMAND.

FOLLOWING IS A LIST OF ERRORS AND THEIR MEANING:

? THAT ROUTINE HAS ALREADY BEEN PASSED
AN ATTEMPT WAS MADE TO ALTER A ROUTINE WHICH HAD ALREADY BEEN COPIED TO THE OUTPUT FILE.

? THAT ROUTINE DOES NOT EXIST
AN ATTEMPT WAS MADE TO CORRECT, DELETE OR REPLACE A ROUTINE NOT IN THE INPUT FILE.

? THAT ROUTINE ALREADY EXISTS
AN ATTEMPT WAS MADE TO INSERT A ROUTINE WHICH WAS FOUND TO EXIST IN THE INPUT FILE.

? THAT LINE HAS ALREADY BEEN PASSED
AN ATTEMPT WAS MADE TO INSERT, DELETE OR REPLACE A SOURCE LINE WHICH HAD ALREADY BEEN COPIED TO THE SCRATCH FILE.

? THAT LINE DOES NOT EXIST
AN ATTEMPT WAS MADE TO REPLACE OR DELETE A LINE NOT FOUND IN THE SOURCE ROUTINE.

? THAT LINE ALREADY EXISTS
AN ATTEMPT WAS MADE TO INSERT A LINE WHICH WAS FOUND TO EXIST IN THE SOURCE ROUTINE.

? IMPROPER LIBRARY-NAME
THE SPECIFIED LIBRARY-NAME IS LONGER THAN 8 CHARACTERS, OR CONTAIN OTHER THAN THE CHARACTERS A-Z, Ø-9 AND HYPHEN.

3.6.3 HARDWARE ERRORS

IF AN ERROR IS DETECTED WHILE READING OR WRITING ON A DEVICE, THE PROGRAM WILL TYPE

```
    ? ERROR ON FILE  DEV:FILE.EXT
```

4. INTERNAL ENVIRONMENT

4.1 TRADE-OFFS

TWO PROGRAMS WILL BE USING THE LIBRARY FILES, THE LIBRARY MAINTENANCE ROUTINE AND THE COBOL COMPILER. SINCE THE LIBRARY FILE HAS A REASONABLY COMPLEX STRUCTURE (ROUGH TABLE, FINE TABLE, ASCII SOURCE LANGUAGE) ONE OF THE TWO PROGRAMS WILL HAVE TO DO SOME FILE ORGANIZATION. IT WOULD SEEM BEST THAT THE MAINTENANCE ROUTINE GO TO SOME EXTRA TROUBLE TO PUT THE OUTPUT IN A FORM EASILY ACCESSIBLE TO THE COBOL COMPILER.

THE USER WILL PROBABLY WANT TO UPDATE ONLY A SMALL PORTION OF THE LIBRARY. THE LIBRARY MAINTENANCE MAY EITHER INSERT A "BULGE" IN THE FILE, OR COPY THE FILE WITH CORRECTIONS. THE LATTER APPROACH WOULD SEEM TO BE PREFERRED, BOTH FOR THE SAKE OF SAFETY AND TO AVOID THE HORRORS OF PLAYING WITH LINKS, POINTERS OR WHAT-NOT.

SINCE THERE IS A TABLE CONTAINING THE NAMES OF THE SOURCE ROUTINES IN THE LIBRARY, IT WOULD BE POSSIBLE TO PUT THOSE ROUTINES IN ANY ORDER. HOWEVER, THE COBOL COMPILER COULD FIND A GIVEN SOURCE ROUTINE MUCH QUICKER IF IT HAD TO SEARCH ONLY ONE BLOCK OF TABLE FOR THE ROUTINE NAME. THIS CAN MOST EASILY BE DONE IF THE SOURCE ROUTINES ARE SEQUENCED ON ROUTINE NAME.

4.2 SOFTWARE INTERFACES

4.2.1 FORMAT OF THE ROUGH TABLE

THE ROUGH TABLE CONSISTS OF A SINGLE BLOCK OF 128 WORDS. THE FIRST WORD IS UNUSED, THE LAST WORD CONTAINS ALL ONES. THE REMAINING SPACE IS DIVIDED INTO 2-WORD ENTRIES:

WORD 1 AND BITS Ø-11 OF WORD 2 CONTAIN THE ROUTINE NAME, IN SIXBIT, THAT CAN BE FOUND IN THE FIRST ENTRY OF A FINE TABLE BLOCK.
WORD 2, BITS 12-28, CONTAINS THE BLOCK NUMBER OF THAT FINE TABLE BLOCK RELATIVE TO THE BEGINNING OF THE FILE.
WORD 2, BITS 29-35 ARE NOT USED.

4.2.2 FORMAT OF THE FINE TABLE

THE FINE TABLE CONSISTS OF ONE OR MORE BLOCKS OF 128 WORDS. DIVIDED INTO 2-WORD ENTRIES:

WORD 1 AND BITS Ø-11 OF WORD 2 CONTAIN THE NAME OF A ROUTINE IN THE LIBRARY, IN SIXBIT.

WORD 2, BITS 12-28 CONTAINS THE BLOCK NUMBER, RELATIVE TO THE BEGINNING OF THE FILE, WHICH CONTAINS THE START OF THE FIRST LINE OF THAT SOURCE ROUTINE.

WORD 2, BITS 29-35, CONTAIN THE RELATIVE WORD NUMBER WITHIN THAT BLOCK WHICH CONTAINS THE SEQUENCE NUMBER OF THE FIRST LINE OF THAT ROUTINE.

4.2.3 FORMAT OF THE SOURCE ROUTINES

SOURCE ROUTINES CONTAIN LINES OF COBOL SOURCE. EACH LINE HAS A LINE-NUMBER WORD, FOLLOWED BY A STRING OF ASCII. THE LINE-NUMBER WORD HAS A LINE, OR SEQUENCE, NUMBER IN BITS Ø-34 AND A ONE-BIT IN BIT POSITION 35. ANY SPACE LEFT BETWEEN THE LAST CHARACTER OF A LINE, AND THE FOLLOWING LINE NUMBER, IS FILLED WITH NULLS.

THE LAST LINE IN A ROUTINE IS FOLLOWED BY A LINE WITH A SEQUENCE NUMBER OF ALL ONES.

4.3 LANGUAGE

THIS IS WRITTEN IN MACRO-1Ø ASSEMBLY LANGUAGE.

5. EXTERNAL ENVIRONMENT

5.1 EXECUTION SPEED

THE RUNNING TIME FOR THE PROGRAM DEPENDS UPON THE SIZE OF THE LIBRAY FILE, AND THE TYPING SPEED OF THE USER.

5.2 USE

THE PROGRAM IS USED TO CREATE OR UPDATE A FILE USED BY THE COBOL COPY VERB.

6. DOCUMENTATION

6.1 MAJOR ASPECTS

FINAL DOCUMENTATION WILL CONSIST OF:

1) MACRO LISTING WITH COMPLETE COMMENTS
2) A DESCRIPTION OF THE COMMAND LANGUAGE
3) A LIST OF OPERATOR ERRORS
4) MAINTENANCE DOCUMENT

6.2 CHECKOUT

ONCE THE PROGRAM HAS BEEN DEBUGGED BY THE IMPLEMENTOR, USING WHATEVER METHODS HE SEES FIT, IT WILL BE USED BY THE GROUP WRITING THE TEST SYSTEM FOR A COMPLETE CHECKOUT.

6.3 MARKETING

THE COPY CLAUSE IN COBOL IS ONE OF THE MORE POWERFUL PROGRAMMING TOOLS, SAVING THE PROGRAMMER TIME IN BOTH WRITING AND DEBUGGING HIS PROGRAM. THIS PROGRAM ALLOWS THE USER TO SPECIFY THOSE SOURCE ROUTINES WHICH ARE USED IN MANY PROGRAMS, AND PLACE THEM IN A COMMON FILE FOR USE BY THE COBOL COMPILER.