2212 5397

DISTRIBUTION LIST

81800/81700 SOFTWARE PRODUCT SPECIFICATIONS

Detroit — **Single Copy**

T. Freeman - Prod. Mgmt.
K. Stokes - Prod.Mgmt
F. Schoeman - International
H. F. Hayde - International
S. Johnson - BMG
W. Varns - BMG
L. Atkins - BMG
B. Dent - CSG
J. Shifman - CSG
J. G. Cleary - SSG
D. Hill - TC, BM & SS
V. Morton - GPS, BM & SS
P. E. Pleming - Int'l F. E.
D. Dahm - Corp. Eng.

U.S. and Europe — **Single Copy**

K. Conry (Plymouth)
D. R. Bookwalter (Plymouth)
J. H. Pedersen (Plymouth)
J. Berta (Downingtown)
W. Minarcik (Paoli)
G. Smolnik (Paoli)
J. Murtaugh (Tredyffrin)
A. Kosla (McLean)
A. Lacaneta - F&SSG (McLean)
B. Bell (Malvern)
Mgr, WADC (Irvine)
R. Solt (Pasadena)
H. M. Townsend (Pasadena)
N. Cass - Pat. Atty. (Pasadena)
E. Sweaney (Mission Viejo)
E. D. Earnest (Mission Viejo)
J. J. Dowling (Westlake)
J. C. Allan (Glenrothes)
W. McKee (Cumbernauld)
I. J. Carradine (Cumbernauld)
Mgr, NPSGrp (Ruislip)
P. R. Evans (Middlesex)
B. Hammersley (Croydon)
J. Gerain (Pantin)
A. Isola (Gennevieliers)
J. Cazanove (Villers)
P. Cornil (Seneffe)
R. Bouvier (Liege)
J. C. Wery (Liege)
S. Samman (Liege)

Santa Barbara Plant — **Single/Multiple**

R. S. Bunker
J. Hale
R. Shobe
K. Meyers
A. van der Linden
T. Cardona
R. Bauerle
L. Thomas
B. Dodson
J. Henige
J. Casey
E. Yardi
J. Darga
B. Ross-Smith
E. Munsch - 2
G. Hammond - 2
K. King - 6

Distribution current as of 2/13/78

PAGE ____ OF ____

CONTROL DOC. None

DOCUMENT TRANSMITTAL

SYSTEM B1800/B1700

PL FILE NO. ____

PROJ. NO. ____ COORD. ____ CHKR. J Vale TO RECORDS 3-8-78 DISTRIBUTION 3 8 78

DOC PFX	DOCUMENT NUMBER	REV	UNIT NUMBER	DL NO.	PL FILE NO.	ENG. RECORDS ONLY QTY REV PAGE	COPIES EACH PAGE	SEPIAS	MT REQ'D	CD DK REQ'D		REMARKS
PS	2212 5397	C				128						

RECEIVED
MAR 14 1978
GENERAL MANAGER
SANTA BARBARA PLANT

DOCUMENT TRANSMITTAL

Burroughs Corporation

COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

B1800/B1700 COBOL COMPILER LOGIC

PRODUCT SPECIFICATION

REV LTR	REVISION ISSUE DATE	APPROVED BY	REVISIONS
A	10/13/76	J Hale	Original Issue
B	9/20/77	J Hale	Changes for the VI.I Release

Page	Change
2-23	Updated TABLE 2.2: Changed FDI.LIST.HEAD from 295 BITS/ENTRY to 359 BITS/ENTRY.
2-45 to 2-57	Added new section, INDEXED I-O IMPLEMENTATION , consisting of: GENERAL description. TAG FILE FORMAT section ROUGH TABLE FORMAT section ADDING RECORDS TO THE FILE section FILE SEARCHING THE TAG FILE section FILE-CONTROL ENTRY section FILE STATUS VALUES section CLOSE STATEMENT section DELETE STATEMENT section. (Not Yet Implemented) OPEN STATEMENT section READ STATEMENT section REWRITE STATEMENT section START STATEMENT section USE STATEMENT section WRITE section
2-60 to 2-72	Added INDEXED FILE MODEL (sample COBOL program)
4-5	Updated FPB INFO (80) Table: Changed UNUSED from 28 to 26.
4-22	Added FILE ACCESS and ORGANIZATION information to DNINFO chart.

REV LTR	REVISION ISSUE DATE	APPROVED BY	REVISIONS
C	3/7/78	J Hale	CHANGES FOR 7.0 RELEASE LEVEL

Page	Change
3-1	Syntax for Data-Base Declaration added
3-2	Added: 02 FILLER BIT(4) and changed BIT(12) for STR.NUMBER to BIT(8). Added to PARSE: "DMCATEGORY becomes a separate TRESWD temporarily." Added to EXPLODE: "For DMCATEGORY, transform to a normal TWORD, allowing direct access by any appropriate COBOL operation."
3-3	Added to: When DB is declared, the compiler: "If present, outputs OF followed by ..." Changed: When Data Set is declared, the compiler: from "IF ORDERING THEN 51 ELSE 52" to "IF 'RETRIEVAL' THEN 52 ELSE 51."

PRODUCT SPECIFICATION

REV LTR	REVISION ISSUE DATE	APPROVED BY	REVISIONS
C			(Continued) 3-6 Added last sentence to Data-Base Section Processing 4-1 "created in EXPLODE" added to 03 TFILEREC 4-4 BASIC COP(53) - SUB FLAG (1) shown as MULTI ENTRY 4-22 CODEGEN will always do a get for a reserved word "unless INHIBIT.GET.OP is set". 4-23 "Used by computer" changed to "Used by CODEGEN"

BURROUGHS CORPORATION
COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

TABLE OF CONTENTS

DESIGN FEATURES

SOURCE LANGUAGE SELECTION

The COBOL Compiler is a multipass compiler which accepts ANSI '68 COBOL with Burroughs extensions. The high level of each language module of USA Standard COBOL, rather than COBOL subsets, was implemented because subset language elements seemed arbitrarily selected, difficult to use, and restrictive. The compiler was written to be compatible with B3500 language extensions rather than geared toward the lower end of the B1700 memory scale.

"Multipass" or "multiphase" compilation better utilizes small memory stores. (Also, it is possible for one phase to be coded and checked out through intermediate files before the final code file is built.) The original source code is transformed to a more convenient form by passing it against a part of the compiler. The transformed text is passed to the next phase via an intermediate work file. In this way, data is managed in a quasi-sequential manner and random overlays are minimized, as in a data paging or a virtual-memory system of management. In the same way, code overlays are minimized because only a part of the code is invoked to transform all the data for the phase.

Multipass compilation does have the following disadvantages which need to be recognized:

-- The format of each token, as it appears to each pass, must be well defined. If token formats change, each instance must be changed.

-- Extra code is required in each phase to get and put tokens.

-- If a large memory is available, there is no easy way to combine phases, e.g., label and data-name qualification resolution could be combined if there was space for the intermediate tables.

-- A certain fixed overhead is required for opening and closing the intermediate files, even for a minimum source language deck.

This product specification will discuss the design features, general functions, file structures, source record merge procedures, and data management processes of the COBOL Compiler. For further information about virtual machine language or about COBOL source language, the user is instructed to see the

appropriate documentation (See RELATED DOCUMENTATION below).

RELATED DOCUMENTATION

NAME	NUMBER
USA Standard COBOL	USAS X3.23-1968
COBOL S-Language	P.S. 2201 6729
COBOL Compiler	P.S. 2212 5314
COBOL Reference Manual	1057197
DMS II	P.S. 2212 5470

DEBUGGING CAPABILITIES

A variety of debugging output has been incorporated in the COBOL compiler in order to minimize the debugging error detection and error correction time. This output is printed when the following dollar card reserved words are used:

<NO> PARSE:	Initial parsing
<NO> DICT:	Dictionary processing
<NO> DNQUAL:	Data-name qualification resolution
<NO> LQUAL:	Label qualification resolution
<NO> MOIGE:	Merge (Note: "MERGE" has another meaning)
<NO> DATSYN:	DATA DIVISION syntax checking
<NO> EXPLODE:	Explode
<NO> PROSYN:	PROCEDURE DIVISION syntax checking
<NO> CODEGEN:	Code generation
<NO> FIXUP:	Fixup the codefile

When this option is used, a formatted token as seen by the GET and PUT procedures for that phase is printed. In addition, the name of each procedure and any pertinent variable is printed upon entering the procedure. Most recursive procedures also monitor their exit points for ease of debugging. It is possible to see each process, for the duration of several cards, by turning monitor on or off for the desired phase(s).

DYNAMIC MEMORY

The COBOL Compiler is designed for many memory configurations, and can effectively utilize additional memory if it is available. The COBOL compiler accomplishes this by managing the dynamic space declared at compile time.

A certain minimum space is required for building lists and tables during the different phases. Associated with this minimum space are limits, e.g., the number of DATA-NAMES, PROCEDURE-NAMES, etc. If a particular source program exceeds any of these limits, more space must be dedicated by increasing the dynamic space and recompiling. An attempt has been made to make these restrictions "reasonable".

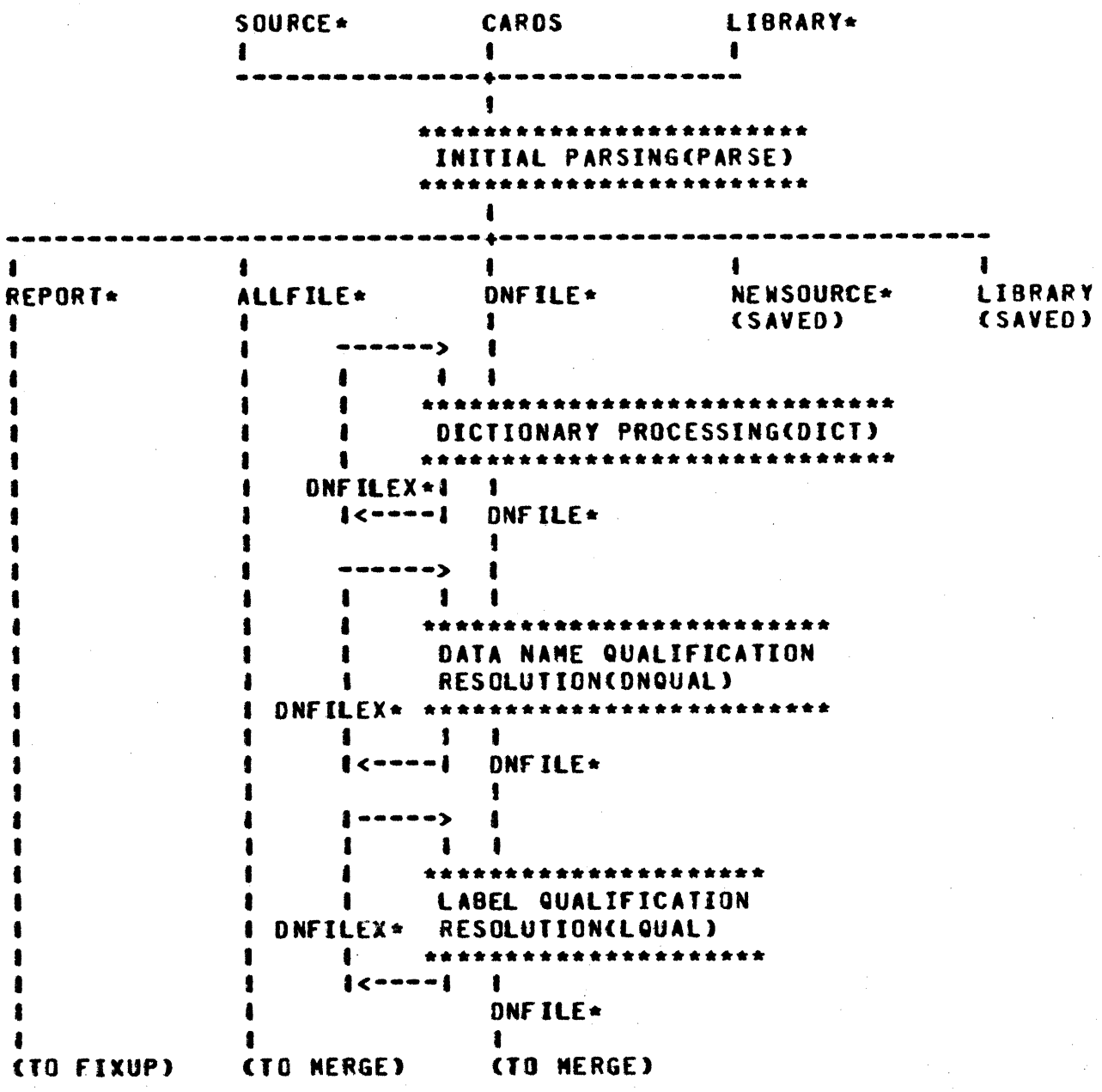
If more space is available, the compiler is designed to use that space for a significant speed gain.

INTERMEDIATE FILE DESIGN

In order to minimize the intermediate file sizes, information about the original text is distributed to several files which are ordered to each other. The individual files are then processed without the need to copy extraneous information. When all of the transformations are complete the files are merged.

Care must be taken to preserve the original ordering of the files.

The diagram below shows the flow of the intermediate files from one phase to another. The first record of the SEGFILE is used to pass global data between phases, e.g. the DONT.GENERATE.CODE flag or the ABORT flag in case of a drastic error, etc. This aspect of the SEGFILE is not shown.



* Indicates workfile names in the form COBOLW/<name>

Table 1.1 Flow of Intermediate Files

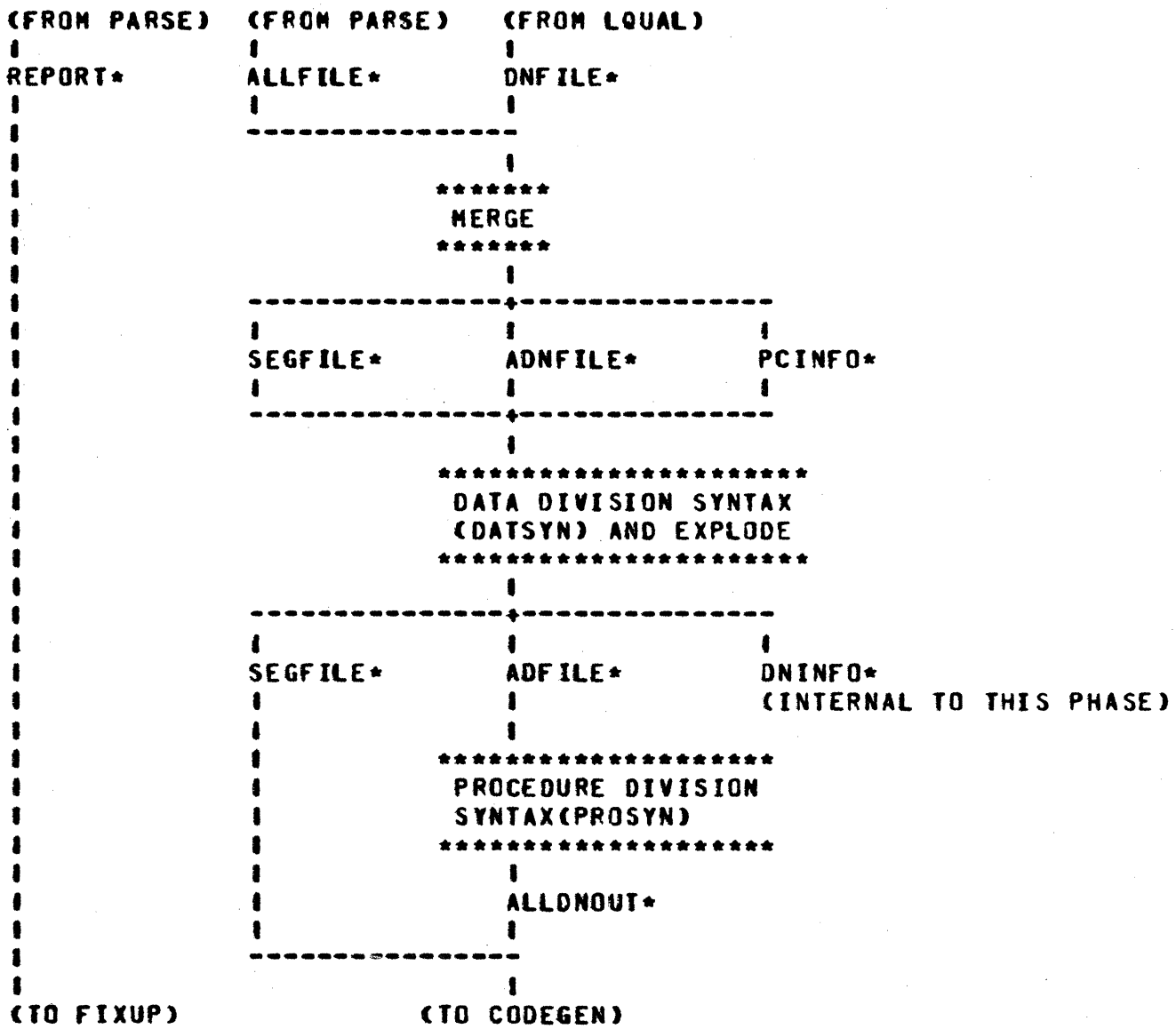


Table 1.1 Flow of Intermediate Files (Cont.)

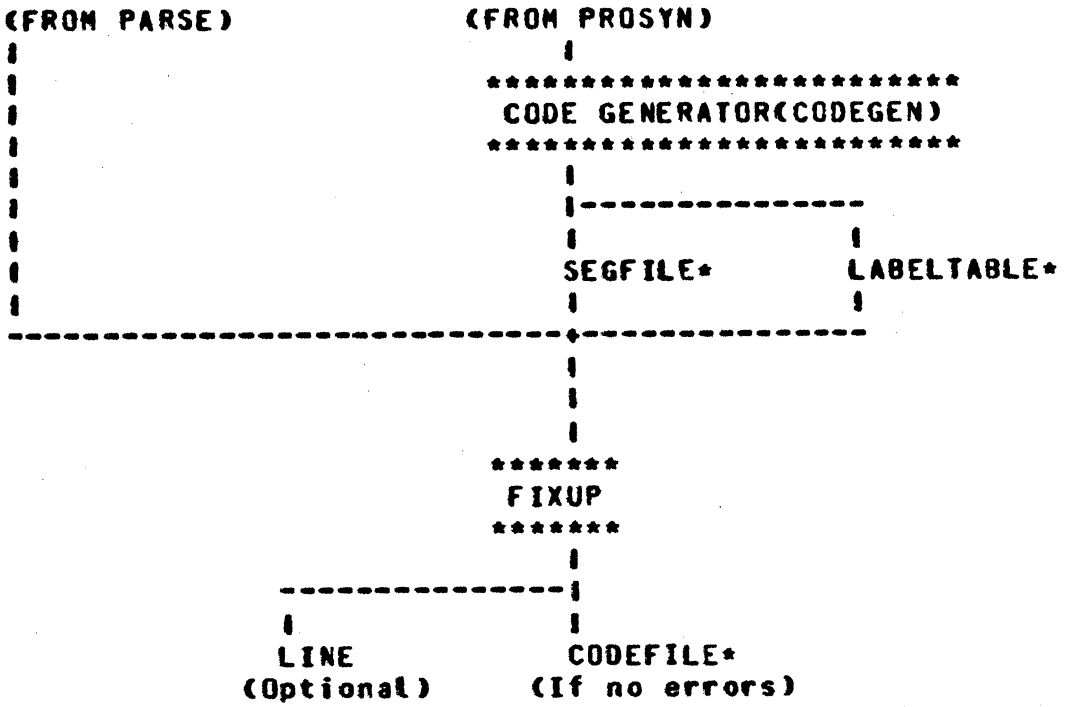


Table 1.1 Flow of Intermediate Files (Cont.)

*NOTE: As of 6.1 Version, in order to reduce the OPEN/CLOSE overhead, each serial workfile is now opened I/O and repositioned as needed. Further, the following logical files no longer exist as physically separate entities:

ALLFILE and DNFILE now use ALLDNOUT
DNFILE and LABELTABLE now use ADFILE
PCINFO now uses DNFILEX.

PHASES OF THE COMPILER

The following is a functional description of each of the compiler phases and its intermediate files. Where applicable, a description of how dynamic memory is managed is included.

INITIAL PARSING (PARSE)

INPUT FILES

CARDS (reader):	Source card images or patch cards.
SOURCE (optional disk):	source card images to which patches may be applied.
LIBRARY (optional disk):	Source card images merged with the primary file when a COPY statement is encountered.

OUTPUT FILES

NEWSOURCE (optional disk):	Source card images to which patch cards may have been applied.
LIBRARY (optional disk):	New library file(s) created when "L" is specified in column 7.
REPORT (disk):	All card images processed including dollar cards, library cards and patch cards. This file is used to print the optional listing or error report.
ALLFILE (disk):	Contains all constant information about each token processed. This becomes the controlling file during the MERGE phase.
DNFILE (disk):	Contains picture strings and all variable names isolated, e.g. section-names, paragraph- names, and data-names.

GENERAL FUNCTIONS

1. Merge source language inputs including library card images.

2. Scan and isolate basic symbols e.g., words, integers, non-numeric literals, numbers, etc.

3. Look up reserved words in a reserved word list.

4. Look up words in the optional COPY REPLACING list or the optional SPECIAL-NAMES list.

5. Verify that all for divisions are present and in the proper order. If this much is not correct, the ABORT flag is set to bypass the regular processing and shortcut to the FIXUP phase.

RESERVED WORD LOOKUP

In order to minimize the memory required to hold them, the COBOL reserved words are organized into two primary lists:

1. IDENTIFICATION DIVISION thru WORKING-STORAGE SECTION reserved words.

2. PROCEDURE DIVISION reserved words.

The words are arranged in sublists by length. For example, "IS", "BY", "OF", etc. appear with other two character reserved words. A guess as to the frequency of use is applied to each sublist with the more frequent words appearing at the beginning of the sublist.

When a match is found by searching the sublist sequentially, the reserved word key and category are provided. Noise words, NOTE sentences, and NOTE paragraphs are deleted when detected.

SPECIAL FUNCTIONS

There are certain special functions accomplished in this phase. These appear below according to the division or section header associated with the function.

MONITOR SENTENCE

1. Verify that a file-name is declared.

2. Identify and mark data-names and procedure-names to be MONITORed.

3. Pass these names to the MERGE phase as non-numeric literals on the ALLFILE.

4. If the word "ALL" appears as the procedure-name list, the name of each section and paragraph is passed to the MERGE phase as a non-numeric literal on the ALLFILE.

DATE-COMPILED

1. Scan to the end of the sentence.

2. Build an image containing date and time and output it to the REPORT file.

OBJECT-COMPUTER

1. Verify and save the MEMORY SIZE value (in characters) for the FIXUP phase.

2. Save SEGMENT-LIMIT value.

3. Save DATA SEGMENT-LIMIT value.

SPECIAL-NAMES

1. Save the CURRENCY SIGN value.

2. Save the DECIMAL-POINT (comma value).

3. Build the mnemonic names list.

DATA DIVISION

1. Assign an OCUR number to each file-name and data-name declared. This number begins with one and increases by one (zero is reserved for error reporting). This OCUR becomes the internal representation for the data-name after the DICT phase. FILLER entries are not assigned an OCUR unless they are 01 FILLER ... entries of a file.

2. Output a dummy FD to correspond to the WORKING-STORAGE SECTION header. This dummy FD is required for the DNQUAL phase.

FILE SECTION

1. Parse the file-name of an FD or SD declaration. Set level to zero and the appropriate flags true.

2. Drop the DATA RECORD(S) clause and its operands.

3. Mark the operands of the LABEL RECORD(S) clause.

4. Parse the item descriptions of the file record (See WORKING-STORAGE SECTION).

WORKING-STORAGE SECTION

1. Parse item descriptions.

2. Isolate and verify proper level indicator.

3. Recognize and mark FILLER entries.

4. Mark the operands of a REDEFINES or RENAMES clause.

5. Recognize a PICTURE declaration and control the scanner to isolate a PC string.

6. Mark index-names.

7. Mark all items that are not "corresponding" candidates, e.g. entries with FILLER specified, level = 66, 77, or 88, and entries containing an OCCURS or REDEFINES clause. These are not considered for selection of corresponding pairs.

PROCEDURE DIVISION

1. Recognize and mark section-names and paragraph-names and assign them a label OCUR number. This number begins with one and increases by one for each new section or paragraph (zero is reserved for error reporting). This label OCUR becomes the internal number for the name after the DICT phase.

2. Assign a segment number to each section based on SEGMENT-LIMIT and priority-number declarations. These segment numbers are secuential starting with zero.

3. Recognize GO TO paragraphs and mark them as legal operands of an ALTER statement.

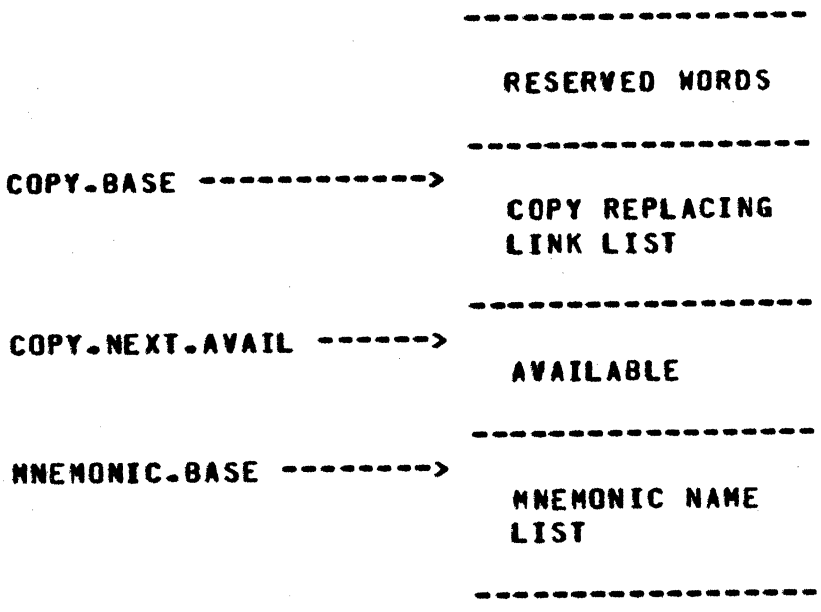
4. Identify and mark operands of a GO TO, ALTER, PERFORM, or a PERFORM ... THRU ... statement.

5. Mark operands of other special I/O verbs as procedure names.

6. Recognize and mark operands of a CORRESPONDING verb (e.g. MOVE CORRESPONDING A TO B).

DYNAMIC MEMORY

The dynamic memory for parse is utilized as follows:



If COPY.NEXT.AVAIL ever meets MNEMONIC.BASE then an error message is issued with a request to recompile using more memory.

DICTIONARY PROCESSING (DICT)

INPUT FILES

DNFILE:	Contains all picture strings, declared variable names and references to the variables.

OUTPUT FILES

DNFILE:	All picture strings and variables are reduced to an OCUR.

GENERAL FUNCTIONS

1. Build a dictionary of declared data-names and procedure-names.

2. Look up references to the declared variables and substitute an OCUR number and a same name OCUR number for the symbol string. The same name OCUR number points to any prior occurrence of this name in a link list fashion. If the same name OCUR = the OCUR then the last element of the list has been reached or the name is unique.

3. Assign a picture OCUR number to each new picture string that was declared. A picture string is distinguished from other variable names by adding a blank to the end of the string and increasing the symbol length by one in the scanning process(PARSE).

4. These functions are done on an iterative basis. When the first attempt is made to add a name to the dictionary and it will not fit, a DICTIONARY.FULL.FLAG is set to one and that token is marked as unprocessed on the output file. When a name is looked up in the dictionary and is found, it is marked as processed on the output file and further processing of that token is inhibited for the duration of the iterations. If the name is not found, and the DICTIONARY.FULL.FLAG is equal to one the token is marked as unprocessed on the output file, otherwise an OCUR number of zero is returned to indicate "UNIDENTIFIER NAME" and the token is marked as processed. This continues until all variable names have been added to the dictionary.

DICTIONARY SEARCH

Associated with each variable name is a stackhead number. This number is created in PARSE by applying a transformation to the character string comprising the name. It is used as an index into a stackhead array to find the beginning of the link list in which this name shall appear. A link of zero indicates end of the list or an empty list.

Two stackhead arrays are maintained in the lookup process, one for data-names and the other for picture strings and labels (section-names and paragraph-names). The picture strings are distinguished from the labels because they have a blank character appended to the end.

DYNAMIC MEMORY

The stackhead arrays and link lists are maintained in the available dynamic memory. One or more iterations through the file may be eliminated by increasing the size of dynamic for this phase.

Each iteration consists of two passes.

PASS I

1. Copy all unprocessed tokens of the ENVIRONMENT DIVISION to the output file.
2. Add all declared data-names and labels to the dictionary.
3. Look up the operands of a REDEFINES or RENAMES clause. This solves the problem of implied qualification for duplicate declarations in that the last mentioned occurrence number will be found rather than the final occurrence of that name.
4. Look up all data-names of the PROCEDURE DIVISION. The OCUR number found will point to the final occurrence of the same name link list.
5. Copy unprocessed labels to the output file.

PASS II

1. Look up unprocessed names of the ENVIRONMENT DIVISION.
2. Copy unprocessed data-names to the output file.
3. Look up label references of the PROCEDURE DIVISION. If the label is a duplicate, the OCUR number of the final declaration will be found.
4. If the DICTIONARY.FULL.FLAG is equal to one then iterate to PASS I.

DATA-NAME QUALIFICATION RESOLUTION (DNQUAL)

INPUT FILES

DNFILE:	Contains a same name OCUR for declared data-names and an unresolved OCUR for data-name references and qualifiers.

OUTPUT FILES

DNFILE:	Contains resolved OCUR for data-name references.

GENERAL FUNCTIONS

1. Starting at the DATA DIVISION of the DNFILE (read only), build an explicit data-name table containing the data-name flags (DN.FLAGS), the same name OCUR (SNAME), and the scope OCUR (SCOPE) for each explicit data-name. The GROUP.FLAG is set at this time.

2. Re-read the entire DNFILE and write the resolved OCUR for referenced data-names. The qualifier tokens are dropped at this time.

3. Set the LABEL.RECORD.OPERAND flag for all elements of a label record.

4. Select corresponding pairs. The pairs are delimited by a corresponding sentinel so that the file can be processed properly in the MERGE phase.

5. Update the data-name reference count based on the resolved OCUR.

6. Set the MONITORed flag of the data-names listed in the MONITOR statement.

7. Read the DNFILE and write a new DNFILE with the updated DN.FLAGS posted to the explicit data-name tokens.

8. When the PROCEDURE DIVISION is found on the input file, begin the LQUAL phase.

SCOPE OF AN ENTRY

The scope of each explicit data-name is calculated from its level number when building the data-name table. The scope of a group item is the last elementary item of the group. The scope of an elementary item or an index-name or a RENAMES entry, or a condition-name is itself.

Example:

```
OCUR  SNAME  SCOPE
----  -----  -----
(01)    01     11     01 A.
(02)    02     08       02 B.
(03)    03     03         88 C VA 1, 5 THRU 10.
(04)    04     04           03 D PC X.
(05)    05     08           03 E.
(06)    06     06             04 F PC X.
(07)    07     07             04 G PC X.
(08)    08     08               88 H VA "2", "3".
(09)    09     09       02 I PC X.
(10)    10     10     66 J RENAMES D THRU F.
(11)    11     11     66 K RENAMES B.
(12)    12     14     01 X.
(13)    02     13       02 B PC X.
(14)    04     14       02 D PC X.
```

The same name and scope attributes are used in resolving references to duplicate names and in the selection of corresponding pairs for a CORRESPONDING verb.

DYNAMIC MEMORY

The explicit data-name table is built in dynamic memory. Each table entry occupies 38 bits. If the entire table can not be contained, an error message is issued with a request to recompile using more memory.

LABEL QUALIFICATION RESOLUTION (LQUAL)

INPUT FILES

DNFILE:	Contains unresolved label references.

OUTPUT FILES

DNFILE:	Contains a unique label OCUR for referenced sections or paragraphs.

GENERAL FUNCTIONS

1. Starting with the PROCEDURE DIVISION of the DNFILE (the file was left open in DNQUAL), build an explicit label table containing the label flags, same name OCUR, segment number, and section OCUR associated with the paragraph if any. The file is written during this process.

2. Read the file and write the resolved OCUR for referenced labels. The label qualifiers are dropped at this time.

3. Mark the MONITORed labels and the terminal paragraph of a PERFORM range.

4. Upon encountering an ALTER operand, verify that it is a GO TO paragraph and mark it as ALTERed.

5. When the end of file is reached, assign a sequential terminal paragraph number to each terminal paragraph of a PERFORM range.

6. Read and write the DNFILE with the updated label information posted to each explicit reference or label reference to it.

DYNAMIC MEMORY

The explicit label table is built in dynamic memory. Each entry occupies 41 bits. If the entire table can not be contained, an error message is issued with a request to recompile using more memory.

MERGE

INPUT FILES

ALLFILE:	Contains constant information about each token processed in PARSE.
DNFILE:	Contains unique OCUR number for each data-name and label referenced.

OUTPUT FILES

ADNFILE:	Contains merged tokens of the ALLFILE and the DNFILE.
PCFILE:	Contains PCINFO entry for each unique picture string of the DATA DIVISION.
SEGFILE:	Edit masks generated by the picture analyzer, and symbols to be printed when MONITORing are added to this file.

GENERAL FUNCTIONS

The primary function of this phase is to merge the ALLFILE and the DNFILE giving the ADNFILE. In addition the following functions are performed:

1. The data-name symbols to be printed when MONITORing are written to the SEGFILE, and the monitored OCUR vs the memory address are put into a table. A table of OCUR numbers vs the label symbols to be printed is placed in another table.

2. Analyze unique picture strings (the picture OCUR of the current PC is greater than the last one analyzed) and create a PCINFO entry on the PCFILE. This entry contains PC attributes like size, scale, class, etc. If an edit mask is required it is written to the SEGFILE and its memory address is posted to the PCINFO entry.

3. A table of condition-name OCUR numbers vs the associated value list is created for each level 88 entry of the DATA DIVISION.

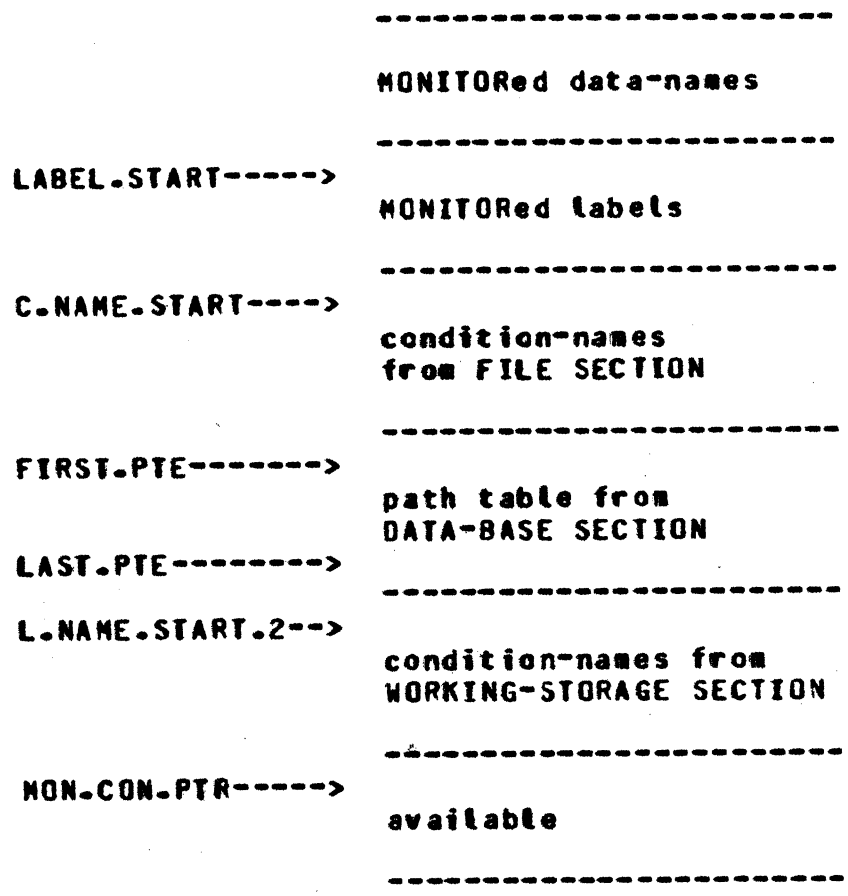
4. When a MONITORed data-name is seen, its OCUR is looked up in the monitor table and the monitor address is posted to that entry.

5. When a MONITOR label list element is encountered, the symbol to be printed is provided as a non-numeric literal following that token.

6. When a condition-name is referenced, the parenthesized text that is equivalent to the desired test is provided from the condition-name table. Note: The parentheses are necessary in the case where the condition-name is negated.

DYNAMIC MEMORY

The MONITOR symbol tables, the path table and the condition-name table are maintained in dynamic memory. If they can not be entirely contained, an error message is issued with a request to recompile using more memory.



NOTE:

The CONDITION-NAME table may become two TABLES FOR PROGRAMS specifying a DATA-BASE SECTION.

Table 2.1 Table Management in Dynamic Memory

DATA DIVISION SYNTAX CHECK (DAISYN)

INPUT FILES

PCFILE: Contains a PCINFO entry for each unique PC string that was declared.

ADNFILE: Processed as a read only file up to the PROCEDURE DIVISION.

OUTPUT FILES

DNINFO: Contains a DNINFO entry for each explicit data-name excluding FILLER entries. This entry contains attributes such as USAGE, address, length, number of subscripts required, BLANK WHEN ZERO, etc.

SEGFILE: Contains DATA DIVISION tokens, e.g., card token with associated address information to be printed, values to which the WORKING-STORAGE variables should be initialized, error or warning messages, etc.

GENERAL FUNCTIONS

1. Perform a detailed syntax check of the source program up to the PROCEDURE DIVISION.

2. Save the file attributes specified in the FILE-CONTROL paragraph. These attributes are held in the FD.INFO table.

3. Save the "SAME RECORD AREA ..." attribute of the I-O-CONTROL paragraph in the FD.INFO table. The multi-file tape id and the multi-file pack id attributes are also saved in the FD.INFO table.

4. Combine the FD or SD attributes declared in the FILE SECTION with those in the FD-INFO table. The set of combined information is used to create the File Parameter Block (FPB) for each file declared.

5. Perform a detailed syntax check of each declared data-name and allocate memory for each entry. A COP index (COPX) is assigned at this time. These attributes, e.g. USAGE, BLANK WHEN ZERO, number of subscripts, address, COPX, etc., are saved in the DNINFO file and can be retrieved by using the data-name OCUR number as the key.

6. Parse the "VALUE IS ..." clause of the WORKING-STORAGE entries. A token for initializing memory at run time is issued to the SEGFILE.

7. Create the pseudo Data Dictionary for data segmentation.

8. When the PROCEDURE DIVISION is found, gather data segments, finalize segment number and displacement, build the COP table, and begin the EXPLODE phase.

DATA SEGMENTATION

When allocating storage for data, a pseudo Data Dictionary entry is built for each data segment candidate, e.g., each file record work area and each WORKING-STORAGE record that is not redefined. The non-contiguous items (level = 77) are assigned to data segment zero. Special consideration is given to these items in that data segment zero is always present and a presence check by the COBOL Interpreter is not necessary.

When the PROCEDURE DIVISION is found, an attempt is made to gather data segments according to the DATA SEGMENT-LIMIT value specified in the OBJECT-COMPUTER paragraph.

If DATA SEGMENT-LIMIT = zero then all data is assigned to data segment zero and no further action is necessary.

If DATA SEGMENT-LIMIT is not = zero then each file record work area is assigned to a new segment and the WORKING-STORAGE records are gathered as follows:

1. If a candidate is greater than DATA SEGMENT-LIMIT then assign it to a new segment.

2. If a candidate will fit with the candidates gathered so far (e.g., the combined size does not exceed DATA SEGMENT-LIMIT) then assign it to the current data segment.

3. If a candidate will not fit in the current segment then begin gathering to a new segment.

This method tends to gather small candidates to a data segment size that is close to the desired DATA SEGMENT-LIMIT.

DYNAMIC MEMORY

Dynamic memory contains several tables which are managed as follows:

1. The PC.TABLE is loaded from the PCFILE into the first part of DYN.WA at the beginning of DATSYN.

2. FDI.LIST.HEAD contains a link to the first file info entry (FDI.INFO) link list. There is one FD.INFO entry per SELECTed file.

3. The Data Dictionary table is built at the end of DYN.WA and goes toward the FD.INFO table. There is one DDICT entry for each data segment candidate. DDICT(0) is reserved for the label record work area of the USE procedures.

4. If DYN.PTR ever meets DDICT.LIMIT then an error message is issued with a request to recompile using more memory.

5. At the beginning of EXPLODE the FD.INFO link list and the Data Dictionary table are replaced by as many DNINFO file entries as will fit. The overflow entries are retrieved on a random basis from the DNINFO file.

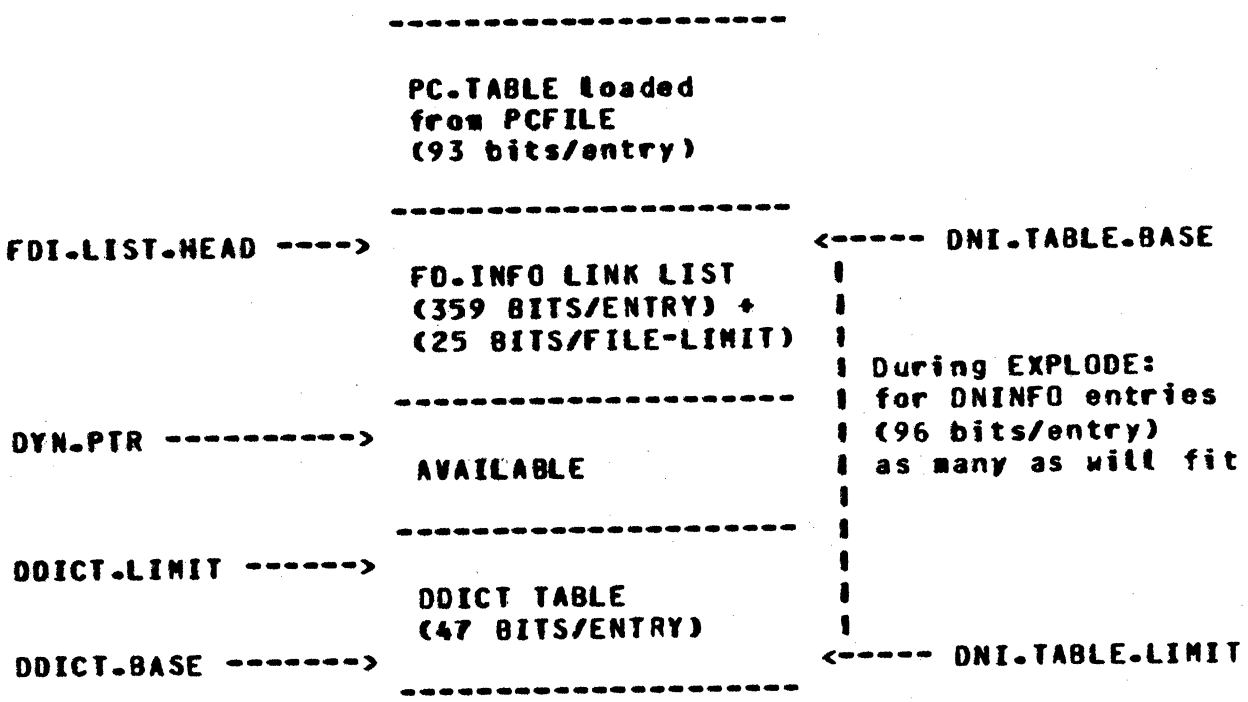


Table 2.2 Table Management in Dynamic Memory

EXPLODE

INPUT FILES

ADNFILE:	Ready to read the PROCEDURE DIVISION.

OUTPUT FILES

ADNFILE:	Token references are replaced by a copy of their attributes.

GENERAL FUNCTIONS

1. Read the DNINFO file and load dynamic memory with as many entries as it will hold.

2. Expand tokens to include all the known attributes for that token: e.g. a data-name OCUR is used to index the DNINFO table, to which the PC attributes are added from the PC.TABLE. Subscript factors and table bound information is issued at this time.

3. Expand special register references to look like data-name references (TALLY, TODAYS-DATE, SW1, ... , SW8, etc.).

4. Output token information about declared FILE-LIMITS.

5. Parse the USE sentence of a USE procedure.

6. Parse SORT statements with particular attention to the "USING..." and "GIVING..." clauses.

PROCEDURE DIVISION SYNTAX CHECKING (PROSYN)

INPUT FILES

ADNFILE: Exploded tokens including a copy of their attributes.

OUTPUT FILES

ADNFILE: Contains tokens that have been syntax checked, rearranged, and simplified for code generation.

GENERAL FUNCTIONS

1. Perform a detailed syntax check of each PROCEDURE DIVISION construct. Any error or warning messages issued are merged with prior messages.

2. Get operands and their subscripts or indexes and automatically stack them for use by each caller. Optimize literal subscripts and indexes when the operand is put to the output file (See "SUBSCRIPT AND INDEX OPTIMIZATION"). The look-ahead feature of the GET procedure can be invoked by setting a flag. In this mode, tokens are presented to the caller one at a time until the flag is reset and normal operation resumes where it left off. This mode is especially useful for optimizing: e.g., the ROUNDED, SIZE ERROR, and multiple receiving field requests are encoded as variations with the arithmetic verbs.

3. Change statements to a simpler form: e.g., corresponding pairs of a MOVE CORRESPONDING statement are put out as separate MOVE statements.

4. Transform arithmetic expressions and Boolean expressions to their parenthesis free polish equivalent by applying the operator precedence rules. An attempt is made to produce equivalent strings for arithmetic expressions and their arithmetic statement counterparts.

5. Supply the implied subject and relational operator to abbreviated conditions.

DYNAMIC MEMORY

The operand stack is maintained in dynamic memory. If it can not be entirely contained, an error message is issued with a request to recompile with more memory.

CODE GENERATOR (CODEGEN)

INPUT FILES

ADNFILE:	Contains the PROCEDURE DIVISION tokens that have been simplified for code generation.

OUTPUT FILES

SEGFILE:	CODEGEN tokens are appended to the tokens produced by the prior phases.
LABELTABLE:	Contains explicit label attributes and implicit label attributes (e.g., branch points of a condition) used for generating the correct branch addresses.

GENERAL FUNCTIONS

1. Check the DONT.GENERATE.CODE flag which is set by any prior phase that detected a syntax error. If it has been set, copy the tokens needed to prepare the error report onto the SEGFILE and begin the FIXUP phase. If no errors have been detected, generate the required code.

2. Generate code for the exponentiate intrinsic. This code is generated only when needed (for certain simple cases the code is emitted in-line).

3. Emit code for FILE-LIMITS checking.

4. Emit code to analyze the reason for a USE procedure being invoked.

5. Generate code for the MONITOR intrinsic.

6. Update the LABELTABLE file for each section, paragraph, and implicit branch that is emitted.

DYNAMIC MEMORY

A GET.TOKEN procedure automatically stacks the operands and their subscripts for use by each caller. The operand stack is maintained in dynamic memory. If it can not be entirely maintained, an error message is issued with a request to recompile with more memory.

FIXUP

INPUT FILES

REPORT:	Contains card images needed for listing. Not opened if no list specified and there are no errors.
SEGFILE:	Contains various information from preceding phases which is used to build the CODEFILE and supply additional data for the listing.
LABELTABLE:	Used to finalize branch addresses. Not present if there are syntax errors.

OUTPUT FILES

CODEFILE:	Contains object program according to MCP specifications. Not present if there are syntax errors.
LINE:	Used for listing and compiler debugging output.

GENERAL FUNCTIONS

FIXUP has two outputs: CODEFILE and a listing. CODEFILE is not produced if there are syntax errors. FIXUP is sensitive to $-CARDS. A default $-CARD is the first token FIXUP sees (as part of the DATSYN tokens in the SEGFILE) unless the user has a $-CARD at the front of his deck. The $-CARD gives "LIST" and "CONTROL" by default.

If the list option is currently off and there is an error or warning, the appropriate source card is printed.

Everything FIXUP does is very specialized and detailed. It knows how the MCP expects a CODEFILE to be formatted, how the COBOL Interpreter wants memory laid out for a COBOL program, and various other housekeeping duties.

HOUSEKEEPING

The COBOL Interpreter permits six different container sizes to be variable length. CODEGEN determines two of these: SEGB (specifies size for the data segment number portion of data addresses) and COPXB (size for COPX). FIXUP determines four of these:

1. BDISPB1 (size for branch addresses + 1).
2. DISPB (size for data address displacement).
3. LENB (size for data length).
4. COPB (size for COP entry := SEGB+DISPB+LENB+4).

FIXUP must also build the ALTER table. The digit size of an entry = (8+BDISPB+3)/4.

FILL DYNAMIC LABEL TABLE AREA

At the beginning of FIXUP, the dynamic area available (if any) is sequentially loaded with as many LABELTABLE entries as will fit, each entry being resolved to its BADDR format (33 bits) on the fly. Overflow entries are retrieved randomly in their original form from the LABELTABLE file and resolved by exception.

BUILD DATA DICTIONARY

FIXUP uses the DNDICTTABLE (built by DATSYN) in the SEGFILE to build the CODEFILE Data Dictionary. Data segment zero size reflects only the users data. Since the MCP expects data segment zero to be a picture of what will be in memory, FIXUP must update DSEG0 to include:

1. EDIT table (8 characters).
2. COP table.
3. Special registers (SW1..SW8, TALLY, DATE, TIME, TODAYS-DATE, TODAYS-TIME, TODAYS-NAME).

4. Constant pool (data-name MONITOR symbols, edit masks, translation tables, FILE-LIMITS).

5. Trash area (intermediate results).

6. ALTER table (if any).

7. Stack.

Data segment portions of the CODEFILE are set to all 0 bits. This is done because upon the first access of a data segment, the MCP reads it off disk if it has been allocated space in the CODEFILE.

BUILD TRANSLATION TABLES

Either the ASCII-TO-EBCDIC and/or the EBCDIC-TO-ASCII table is created only if appropriate translation instructions are generated. The table(s) is located as the last part of the constant pool.

PROCESS MERGE TOKENS FROM SEGFILE

The only valid token is TMEMVALUE. This is for initializing the constant pool portion of the CODEFILE to MONITOR data-name symbols and edit masks.

BUILD CODE DICTIONARY

The Pseudo Code Dictionary (PCD) produced by CODEGEN contains a summary of static code and number of variable length containers for each logical program segment. By now FIXUP has calculated the container sizes and multiplies these by the PCD numbers and adds the PCD static core to build the final Code Dictionary in the CODEFILE.

In addition, "marker" records are written on the SEGFILE. These tell the beginning CODEFILE disk segment address for each logical program segment.

PROCESS DAISYN TOKENS

The REPORT file is used to produce a listing (if applicable) and TCARDADR tokens are fetched to apply data addresses to the listing. TMEMVALUE tokens are processed to implement the "VALUE IS" clause.

PROCESS CODEGEN TOKENS

In order to align printed code and code addresses on the listing, CODEGEN tokens are processed serially. The user may have code segmentation mixed so that pieces of a logical program segment are scattered throughout the program.

FIXUP emits code for the current program segment. If a new segment occurs, FIXUP uses the "marker" records to remember where the code for a segment left off on the CODEFILE.

The LABELTABLE file is accessed whenever a TBADDR token is encountered. The routine LABEL-FIXUP takes a LABELTABLE OCUR from the token and uses it as an index into the LABELTABLE to finalize a branch address.

FILE-LIMITS values are set up by CODEGEN as TMEMVALUE tokens and are initialized on the CODEFILE at this point.

CLEANUP

1. Print the Code Dictionary.
2. Build the File Parameter Blocks.
3. Print the Data Dictionary.
4. Build and print the Path Dictionary (Data Management).
5. Build and print the COP table.
6. Build and print the run structure.
7. Finalize the Program Parameter Block.
8. Build and print the ALTER table.

9. Print the Program Parameter Block.

10. Print the summary information.

MONITOR STATEMENT

The MONITOR statement must precede IDENTIFICATION DIVISION.

Syntax MONITOR <file-name> [DEPENDING] (<data-name list> : <label list>).

The <label list> may consist of only "ALL" which implies every program label.

Symbols for MONITORed data-names are put in data segment zero. This is because it is likely that a data-name will be a receiving field more than once and this method saves carrying the symbols as a literal in the code segments.

Symbols for MONITORed program labels are set up as literals in the code where the label explicitly occurs.

A DATSYN routine (BUILD.COPS) is responsible for assigning a monitor buffer of 132 characters in data segment zero. It also builds a monitor COP as COP[1] unless "NOCOP" is specified.

At code segment zero, displacement zero, a routine is emitted which consists of a MVN, COMMUNICATE, and XIT. This achieves a write on the user specified line printer from the monitor buffer. The routine is entered after the monitor buffer has been set up at various points in the program. Motivation for the routine is to save code space.

MONITOR TOKENS

e.g: MONITOR PRINT (MASTER, XXX : PAR1).

MERGE INPUT		MERGE OUTPUT	
MONITOR	(TRESWD)	MONITOR	(TRESWD)
PRINT	(TWORD)	PRINT	(TWORD)
MASTER	(TWMON)	.	
"MASTER"	(TNNLIT)	.	
XXX	(TWMON)	.	
"XXX"	(TNNLIT)	.	
PAR1	(TLMON)	.	

"PAR1" (TNNLIT)

PAR1 (TXPAR) where the LABEL appears
"PAR1" (TNNLIT)
.
MASTER (TWORD) with MONITOR LENGTH and
MONITOR SYMBOL address

DATSYN OUTPUT

MONITOR (TRESWD)
PRINT (TFILEREC) with MONITOR buffer address
PERIOD (TDOT)

e.g: MONITOR DEPENDING PRINT (A, B, C : ALL).

MERGE INPUT

```
MONITOR (TRESWD)
DEPEND  (TRESWD)
PRINT   (TWORD)
A       (TWMON)
"A"     (TNNLIT)
B       (TWMON)
"B"     (TNNLIT)
C       (TWMON)
"C"     (TNNLIT)
```

No indication of "ALL" because TNNLITs appear after each explicit LABEL.

SUBSCRIPT AND INDEX OPTIMIZATION

RULES FOR SUBSCRIPTS

1. Literal subscripts cannot be signed. Therefore, they can not generate too low an address.

2. The bound is the maximum digit displacement of the last valid entry (which is any length and may be in digits or characters).

3. The interpreter checks each subscript for less than or equal zero; if true, it generates an error. After all gathering and multiplying of subscripts is done, this value is checked against bounds and if less than the bounds, it generates an error.

4. If all subscripts for a data-name are literals, PROSYN checks for bounds errors. Otherwise, even though a literal may be larger than the corresponding OCCURS value in the DATA DIVISION declaration, it is not checked (because, for example, the value of another subscript may offset this value and bring the total within bounds).

RULES FOR INDEXES

1. Data-name(index + literal) : The data-name address is optimized and the literal dropped.

2. Data-name(index - literal) : If the negative literal plus the data-name address are greater than zero, it is optimized; otherwise, it is not, since it is not known if the index will make the final address within bounds. Errors of this nature are normally caught by the interpreter.

GLOBAL-DOLLAR FORMAT

BIT	RESERVED WORD FOR $ CARD	
0	LIST	
1	DOUBLE	1=DOUBLE, 0=SINGLE (DEFAULT)
2	SUPPRESS	
3	SPEC	
4	CONTROL	
5	CODE	
6	ANSI	(vs. default B3500 COBOL which associates "ELSE" with SIZE ERROR clause and normal I/O AT END and INVALID KEY)
7	PARSE	(MONITOR)
8	DICT	(MONITOR)
9	DNQUAL	(MONITOR)
10	LQUAL	(MONITOR)
11	NOIGE	(MONITOR)
12	DATSYN	(MONITOR)
13	EXPLODE	(MONITOR)
14	PROSYN	(MONITOR)
15	CODEGEN	(MONITOR)
16	FIXUP	(MONITOR)
17	DEBUG	(NO DEBUG yields no code for COBOL MONITOR or DUMP...DEBUG is default)
18	ERRMESS	(gives listing of all COBOL errors)
19	NOCOP	
20	REF	(produces MONITOR code for variables in COBOL MONITOR list and referenced as other than receiving field... e.g: IF A=8.)
21	TIME	
22	EXAMINE	causes S-code to be generated for the EXAMINE verb instead of the S-operator.
23	LISTP	(pre-listing and errors printed as found per phase)
24 - 28	UNDEFINED	
29	RESERVED	

USE OF DYNAMIC

I. PARSE
   A. RESERVED WORDS: approx. 16500 bits
   B. COPY ... REPLACING A BY B...
      1. A requires 14+(8*Length of A) bits.
      2. B requires:
         a. When B is a word 14+(8*Length of B) bits.
         b. When B is a qualified word.
            14+(8*length B)+
            14 + (8*Length of each qualifier) +
            22 for each IN/OF.
         c. When B is a literal
            22+(8*length of B).
      3. Links require 24 bits per replacing entry (i.e., each A BY B).
   C. MNEMONIC NAMES
      24+(8*length of MNEMONIC symbol) for each
      +16 bits for an end of list marker

II. DATA NAME QUALIFICATION
Each explicit data-name requires 38 bits (maximum of 16383 entries).

III. LABEL QUALIFICATION
Each explicit paragraph/section name requires 41 bits (max of 16383 entries).
Maximum of 150 ALTERed GO TO paragraphs.

IV. MERGE
   A. Condition names expansion.
      Each 88 VALUE specified requires 33 bits in addition to the length of the VALUE character string itself.
      Add 53 bits overhead.
   B. MONITOR FUNCTION.
      Each data-name monitored requires 44 bits.
      Each label monitored requires 20 bits in addition to the character string of the label itself.

V. DATSYN
   A. PICTURES.
      Each unique PICTURE requires 93 bits.
   B. FD'S.
      Each FD requires 295 bits.
   C. DATA SEGMENTATION.
      Each group of declarations (file record work areas, 77'S, 01'S) which are candidates for a data segment requires 47 bits for temporary pseudo Data Dictionary.

INITIALIZING GOPAR VALUES OF AN INDEPENDENT SEGMENT

Associated with each independent segment which contains an ALTERed GO TO paragraph, we will define a "SYSTEM" GO TO paragraph.

Associated with each such independent segment at the beginning or end of the physical segment will be a set of ALTER statements to initialize the ALTERed GO TO paragraphs of the segment to their declared procedure-names.

When transferring control to such an independent segment (GO TO or PERFORM), the "SYSTEM" GO TO paragraph is ALTERed to proceed to the desired procedure-name and control is passed to the initializing code for that segment.

Note: A GO TO paragraph contained in a section whose priority is greater than 49 must not be ALTERed from a section with a different priority. (Sections of priority > 49 have all their ALTERs initialized when invoked.)

The following example shows the code generated for various branches to an independent segment which contains ALTERed GO TO paragraphs.

```
FIXED SECTION.

P1.
     PERFORM I2.
          ALTR(I0,I2)
          PERF (K1,OVERLAY)
     GO TO I3.
          ALTR(I0,I3)
          BUN (OVERLAY)
P2.
     GO TO P4.
          GPAR(P2)
P3.
     GO TO.
          GPAR(P3)
       C. COMM(ERROR)
P4.
     ALTER P3 TO I3.
          ALTR(P3,+A)
          BUN(+B)
       A. ALTR(I0,I3)
          BUN(OVERLAY)
     GO TO P1,P3,I7 DEPENDING ON X.
       B. GOTO(COPX,3,+D,P1,P4,+C)
       C. ALTR(I0,I7)
          BUN (OVERLAY)
       D.

OVERLAY SECTION (50).

          ALTR(I4,I6)
          ALTR(I6+E)
          GPAR(I0)

I1.
     ALTER I4 TO I6.
          ALTR(I4,I6)
     ALTER I6 TO I8.
          ALTR(I6,I8)
I2.
          PXIT(K1)
I3.
I4.
     GO TO I6.
          GPAR(I4)
I5.
I6.
     GO TO.
          GPAR(I6)
       E. COMM(ERROR)
I7.
I8.
          BUN(+NI)
```

INDEXED I-O IMPLEMENTATION

GENERAL

Indexed files consist of two physical files:

(1) The data file - contains records written to the file and is maintained in the order in which the records are written.

(2) The tag file - contains the record key value and the relative record number of its associated record in the data file. This file is ordered on the record key value.

TAG FILE FORMAT

The tag file record contains a key and a record pointer. The keys themselves are exact copies of the key field in the data file. The file is blocked in order to fit as many file records into 180 bytes as possible. The number of areas allocated is equal to the number of areas allocated to the data file.

Notes:

1. If the tag file record is greater than 180 bytes then records per block will be set to one (1).
2. Default blocking will be specified for the tag file.
3. Duplicate keys or alternate keys are not allowed.
4. The tag file may not be a multi-pack file.

NEW CONTROL CARD OPTION

Syntax:

```
$ [NO] RPGTAGS
```

Function:

The option must be specified prior to the FD of the indexed file requesting this option and will remain in effect until changed. The option can be reset or set by subsequent control cards.

Notes:

1. If RPGTAGS is specified then

   a) The record pointer size is 6 digits.

   b) The tag file naming convention used for RPG indexed files is used.

2. If NO RPGTAGS is specified then

   a) The record pointer size is 8 digits.

   b) The COBOL tag file naming convention will be used (See tag file name).

3. The default for this $ option is NO RPGTAGS.

TAG FILE NAME

The word "TAG" is prefixed to the internal and external names of the tag file as follows:

DATA FILE INTERNAL FILE NAME	TAG FILE INTERNAL FILE NAME	DATA FILE EXTERNAL FILE NAME	TAG FILE EXTERNAL NAME (DEFAULT)	TAG FILE EXTERNAL NAME($RPGTAGS)
FILE1	TAGFILE1	A/B	TAGA/B	A/TAGB
FILE2	TAGFILE2	FILE2	TAGFILE2	FILE2/TAG
FILE3	TAGFILE3	CCC/A/B	CCC/TAGA/B	CCC/A/TAGB
FILE4	TAGFILE4	A/ABCDEFGHI	TAGA/ABCDEFGHI	A/TAGABCDEFG
FILE5	TAGFILE5	A/B/	A/TAGB/	A/B/TAG

ROUGH TABLE FORMAT

The Rough Table will be built for all open statements except open of an output sequential access file and a sequential input file for which there is no start statement.

1. The default size of the Rough Table is ten entries and may be overridden by a VALUE OF clause in the FD entry as follows:

   ```
   FD FILE-NAME

   VALUE OF CORE-INDEX IS INTEGER CHARACTERS ...  .
   ```

2. The table consists of as many entries as will fit into the space allocated for the Rough Table.

3. An entry consists of that record key which occupies the last record of the file's partition.

EXAMPLE:

ENTRY #	CONTENTS
1	RECORD-KEY(1*P)
2	RECORD-KEY(2*P)
3	RECORD-KEY(3*P)
.	.
.	.
.	.
N	RECORD-KEY(N*P)

Where P = <EOF-Pointer> / <# of Rough Table Entries>

Notes:

1. If P = 0 then set P = 1 --in this case the table is big enough to contain all of the keys of the file and the unused table entries are not included in the search.

2. The Rough Table entries will be sequence checked as the table is built in order to insure that the binary search algorithm will work -- if the table entries are not in sequence, the program will be terminated.

ADDING RECORDS TO THE FILE

When the file is opened, its end-of-file pointer (EOF-POINTER) is obtained and NEW-EOF-POINTER is set to it. New records are appended to the end of each file by adding one to NEW-EOF-POINTER and using that as the actual key for both the data file and the tag file.

Note: When the file is closed or when a START statement is executed for the file, and records have been added to the file, then the tag file will be sorted.

SEARCHING THE TAG FILE

A binary search is used to locate the required key between successive entries in the Rough Table. The partition boundaries are then calculated by multiplying these entry numbers by partition-size. A binary search of the tag file is used to find the required key. If the key is not found, the added records or "add-on's" are serially search for a "hit".

In an effort to minimize the number of disk reads to locate a particular record, the following optimizations are used:

1. The key is checked for equal against the one found in the Rough Table. If it is equal, the tag record is read directly and the binary search of the partition is not necessary.

2. If the partition found in the Rough Table contains the EOF-RECORD, the argument is checked against the key of the EOF-RECORD. If it is greater, the add-on's are searched and the binary search of this partition is not necessary.

3. When records are added to the file, the highest - and lowest - key added are saved. The required key is range checked against these values. If it is outside the range, the serial search of the add-on's is not necessary.

Conclusion:

When records are to be added to the file, they should be added beyond the highest key of the file and they should be ordered.

FILE-CONTROL ENTRY

Syntax:

```
SELECT <file-name>
  ASSIGN TO <hardware-name>
 [RESERVE integer-1 ALTERNATE AREA/AREAS]
 [ORGANIZATION IS SEQUENTIAL/INDEXED]
 [ACCESS MODE IS SEQUENTIAL/RANDOM/DYNAMIC]
  RECORD KEY IS data-name-1
 [FILE STATUS IS data-name-2]
```

1. Verify that RECORD KEY is specified for an indexed file.

2. Mark organization as INDEXED if specified.

3. If DYNAMIC is specified, verify that organization is INDEXED and set FDI.ACCESS = 3.

4. Verify that data-name-1 is of category alphanumeric and is within a record description entry of the selected file.

5. Verify data-name-2 to be a two-character alphanumeric item.

6. Save the RECORD KEY occur and the FILE STATUS occur in the FDI.TABLE.

FILE STATUS VALUES

The following values for a file status data item have been established by the COBOL standard:

00 : A.O.K.

10 : AT END

21 : Sequence error encountered on a WRITE or the RECORD KEY value on a REWRITE was changed since the READ

22 : Duplicate key on WRITE would be created

23 : No record found when accessing this key

30 : Parity error

93 : Mismatch of key in tag file and data file.

CLOSE STATEMENT

Syntax: No change (options specified also apply to the tag file).

Function:

1. Close the data file and mark it as closed
2. Move 00 to file status
3. Close the tag file
4. If PURGE is not specified and additions were made then sort the added keys and merge them with the existing tag file.

DELETE STATEMENT

Syntax:

```
DELETE <file-name> RECORD
 [INVALID KEY <imperative-statement>].
```

1. Verify that INVALID KEY is not specified for a file in SEQUENTIAL ACCESS mode.
2. Verify that INVALID KEY is specified for a file for which there is no applicable USE Procedure.

Function:

This verb will not be implemented until a way of marking the deleted data record is found which does not conflict with the RPG usage of the file.

OPEN STATEMENT

Syntax: OPEN O-I is not allowed.

Function:

1. Move 00 to file status.

2. Open the tag file and the data file and set OPEN-TYPE for subsequent checking in other verbs.

OPEN-TYPE

1 : Opened input
2 : Opened output
3 : Opened I-O

3. For OUTPUT SEQUENTIAL files initialize the last key value to all 2003. for subsequent sequence checking of the file as it is created.

4. For INPUT or I-O files build the Rough Table from the tag file-- this may be omitted for sequential files not referenced by a START statement.

5. For INPUT SEQUENTIAL files set the current record pointer to the first available record of the file and if a REWRITE is specified for this file then clear LAST-I-O-WAS-A-READ.

6. For INPUT or I-O DYNAMIC files set the current record pointer to the first available record of this file.

Note:

For OUTPUT SEQUENTIAL files the tag file and the data file will be SEQUENTIAL access -- in all other cases they will be RANDOM access.

READ STATEMENT

Syntax:

FORMAT 1:

```
READ <file-name> [NEXT] RECORD [INTO <identifier>]
  [AT END <imperative-statement>]
```

FORMAT 2:

```
READ <file-name> RECORD [INTO <identifier>]
  [KEY IS <data-name>]
  [INVALID KEY <imperative-statement>]
```

1. Verify data-name to be that which was specified in RECORD KEY clause for this file (note that this phrase is redundant because alternate keys are not allowed).

2. Verify that NEXT is specified for SEQUENTIAL or DYNAMIC files only.

3. Verify that INVALID KEY or AT END is specified if no applicable USE Procedure is specified.

Function:

1. Move 00 to file status.

2. For FORMAT 1 READ (Sequential Accessing):

   a. If any records have been added to the file since it was opened then sort the tag file.

   b. Set the current record pointer to the next available record in the file.

   c. If EOF is encountered move 10 to file status, and execute the AT END statement or the applicable USE Procedure.

   d. If EOF is not encountered, then read the record. If a REWRITE is specified for this file, then set LAST-I-O-WAS-A-READ and save the record key value for subsequent checking in REWRITE. If the keys do not match, move 93 to file status and execute the INVALID

KEY statement or the applicable USE Procedure.

3. For a FORMAT 2 READ (Random Accessing):

   a. Using the record key as an argument, search the tag file for an equal value.

   b. If found then set the current record pointer to the corresponding record and supply that record in the record work area and verify that the proper record was read to insure file integrity. If the keys do not match move 93 to file status and execute the INVALID KEY statement or the applicable USE Procedure.

   c. If not found move 23 to file status and execute the INVALID KEY statement or the applicable USE Procedure.

REWRITE STATEMENT

Syntax:

```
REWRITE <record-name> [FROM <identifier>]
        [INVALID KEY <imperative-statement>]
```

1. Verify <record-name> to be in an indexed file.

2. Verify that the INVALID KEY phrase is specified if no applicable USE Procedure is specified.

Function:

1. Verify that the file was opened I-O.

2. Move 00 to file status.

3. For SEQUENTIAL files:

   a. If LAST-I-O-WAS-A-READ and the record key value equals that of the last key read then write the record and clear LAST-I-O-WAS-A-READ.

   b. If not LAST-I-O-WAS-A-READ or the record key value is not equal to that of the last record read, then move 21 to file status and execute the INVALID KEY statement or the applicable USE Procedure.

4. For RANDOM and DYNAMIC files:

   a. If the record key is not equal to any record key of the file, then move 23 to file status and execute the INVALID KEY statement or the applicable USE Procedure.

   b. If the record key is equal to a record of the file, then write the record.

START STATEMENT

Syntax:

```
START <file-name> [KEY "="/">"/NOT "<" <data-name>]
      [INVALID KEY <imperative-statement>]
```

1. Verify that file-name is an indexed file with SEQUENTIAL or DYNAMIC access.

2. Verify that the INVALID KEY phrase is specified, if no applicable USE Procedure is specified.

3. Verify that data-name is that which was specified in the RECORD KEY clause for this file.

4. If the KEY clause is not specified, then "=" is the assumed operator and data-name is the record key.

Function:

1. Verify that the file is not opened output.

2. If the file is not SEQUENTIAL, and records have been added since the file was opened, then sort the tag file.

3. Move 00 to the file status.

4. Clear LAST-I-O-WAS-A-READ for a SEQUENTIAL file for which a REWRITE is specified.

5. Comparison of unequal length items presumes the longer field to be truncated to make them equal length.

6. The current record pointer is positioned to the first logical record in the file whose key satisfies the comparison.

7. If the comparison is not satisfied by any record in the file, then move 23 to file status and execute the invalid key statement, or the applicable USE Procedure.

USE STATEMENT

Syntax:

```
USE AFTER STANDARD ERROR/EXCEPTION PROCEDURE
ON INPUT/OUTPUT/I-O/<file-name-1> [<file-name-2>] ... .
```

Function:

1. The designated procedures are executed after completing the standard I-O error routine, or when an AT END or INVALID KEY condition occurs and the phrase is not specified for the I-O statement.

WRITE STATEMENT

Syntax:

```
WRITE <record-name> [FROM <identifier>]
    [INVALID KEY <imperative statement>]
```

1. Verify that the INVALID KEY phrase is specified if no applicable USE Procedure is specified.

Function:

1. Move 00 to file status.
2. For SEQUENTIAL files:
   a. Verify that the file is not opened I-O
   b. if the record key value is greater than that of the previous record written, then write this record.
   c. If the record key value is not greater than the value of the previous record written, then move 21 to file status, and execute the INVALID KEY statement or the applicable USE Procedure.
3. If the record key value is not equal to that of any record in the file, then write the record.

4. If the record key value is equal to that of a record in the file then move 22 to file status and execute the INVALID KEY statement or the applicable USE Procedure.

INDEXED FILE MODEL

The following COBOL program was written in order to simulate the functions required to process indexed files. This program is the model for the code generated by the compiler of each of those functions. All occurrences of $RPGTAGS start in column 73 on a compiler listing. They are shown as X$RPGTAGS in the example due to space restrictions.

```
000050$SUPPRESS
000060$MERGE
000100 MONITOR PRINT
000200      (
000300      :
000400      ).
000450*****************************************************************
000455*                                                               *
000460*       NOTE -  IF $RPGTAGS SPECIFIED, ITEMS SO FLAGGED         *
000465*       IN CC 73 ARE 6 DIGITS LONG INSTEAD OF 8. HOKAY?         *
000470*                                                               *
000475*****************************************************************
000500 IDENTIFICATION DIVISION.
000600 ENVIRONMENT DIVISION.
000700 CONFIGURATION SECTION.
000800 INPUT-OUTPUT SECTION.
000900 FILE-CONTROL.
001000      SELECT PRINT ASSIGN TO PRINTER.
001100      SELECT TAGFILE ASSIGN TO DISK
001200        ACCESS MODE IS RANDOM
001300        ACTUAL KEY IS T-REC-NO.
001400      SELECT DATAFILE ASSIGN TO DISK
001500        ACCESS MODE IS RANDOM
001600        ACTUAL KEY IS D-REC-NO.
001700      SELECT S-TAGFILE ASSIGN TO DISK.
001800      SELECT S-DATAFILE ASSIGN TO DISK.
001900      SELECT SD-TAGFILE ASSIGN TO SORT DISK.
002000 I-O-CONTROL.
002100 DATA DIVISION.
002200 FILE SECTION.
002300 FD PRINT LABEL RECORDS ARE OMITTED.
002400 01 LINE PIC X(132).
002500 FD DATAFILE
002600      FILE CONTAINS 9000 RECORDS
002700      BLOCK CONTAINS 18 RECORDS
002800      VA OF ID IS DATAFILE-ID.
002900 01 D-REC.
003000    02 D-KEY.
003100      03 D-KEY-X PIC X.
003200      03 D-KEY-9 PIC 9(4).
```

```
003300   02 FILLER PIC X(5).
003400 FD TAGFILE
003500      FILE CONTAINS 11000 RECORDS
003600      BLOCK CONTAINS 20 RECORDS
003700      VA OF ID IS TAGFILE-ID.
003800 01 T-REC.
003900   02 T-KEY PIC X(5).
004000   02 T-INDEX PIC 9(8) COMP.                          %$RPGTAGS
004100 01 T-REC-COMP PIC 9(18) COMP.
004200 FD S-TAGFILE
004300      FILE CONTAINS 11000 RECORDS
004400      BLOCK CONTAINS 20 RECORDS
004500      VA OF ID IS TAGFILE-ID.
004600 01 S-T-REC.
004700   02 S-T-KEY PIC X(5).
004800   02 S-T-INDEX PIC 9(8) COMP.                        %$RPGTAGS
004900 FD S-DATAFILE
005000      FILE CONTAINS 9000 RECORDS
005100      BLOCK CONTAINS 18 RECORDS
005200      VA OF ID IS DATAFILE-ID.
005300 01 S-D-REC.
005400   02 S-D-KEY.
005500      03 S-D-KEY-X PIC X.
005600      03 S-D-KEY-9 PIC 9(4).
005700   02 FILLER PIC X(5).
005800 SD SD-TAGFILE
005900      FILE CONTAINS 11000 RECORDS
006000      BLOCK CONTAINS 20 RECORDS.
006100 01 SD-REC.
006200   02 SD-KEY PIC X(5).
006300   02 SD-INDEX PIC 9(8) COMP.                         %$RPGTAGS
006400 WORKING-STORAGE SECTION.
006450 01 DUMMY.
006455                                          03 DONT-COUNT-THIS PC 9 COMP.
006500 03 OPEN-TYPE          PIC 9 COMP.
006600   88 OPENED-INPUT       VA 1.
006700   88 OPENED-OUTPUT      VA 2.
006800   88 OPENED-I-O         VA 3.
006900 03 HALF               PIC 9(10) COMP.
007000 03 AVERAGE REDEFINES HALF COMP.
007050   05 FILLER           PIC 9.
007100   05 T-REC-NO         PIC 9(8).
007200   05 FILLER           PIC 9.
007250*
007300*  THE FOLLOWING IS NEEDED ONLY FOR SEQUENTIAL FILES
007400*    FOR WHICH A REWRITE IS SPECIFIED,
007450*    BUT IS INCLUDED IN STORAGE ALLOCATED ANYWAY.
007500*
007600 03 LAST-I-O-WAS-A-READ-V PIC 9 COMP.
007700   88 LAST-I-O-WAS-A-READ VA 1.
007750*
007800*  IN ACTUAL CODE, TAG RECORD AREA STARTS HERE WITH TAG-KEY;
007850*       (TINDEX IS NEEDED ONLY IN MODEL.)
007870*
```

```
007900 01 D-REC-NO          PIC 9(8) COMP.
007920*
007950*     THE FOLLOWING IS NEEDED ONLY FOR SEQUENTIAL FILE
008000*     OR DYNAMIC FILE  WITH READ...NEXT:
008020*
008050 01 PRIOR-KEY         PIC X(5).
008100 01 PRIOR-KEY-COMP REDEFINES PRIOR-KEY PIC 9(10) COMP.
008200*
008250*     THE FOLLOWING ARE NOT REQUIRED FOR SEQ FILES WITHOUT START.
008300*
008400 01 HI                PIC 9(8) COMP.                      X$RPGTAGS
008500 01 LO                PIC 9(8) COMP.                      X$RPGTAGS
008600 01 RT-MAX            PIC 9(8) COMP.                      X$RPGTAGS
008700 01 RT-INCREMENT      PIC 9(8) COMP.                      X$RPGTAGS
008800 01 EOF-POINTER       PIC 9(8) COMP.                      X$RPGTAGS
008900 01 NEW-EOF-POINTER PIC 9(8) COMP.                       X$RPGTAGS
008950 01 SAVE-T-REC-NO     PIC 9(8) COMP.                      X$RPGTAGS
009000 01 ARGUMENT          PIC X(5).
009100 01 EOF-KEY           PIC X(5).
009200 01 KEYS-ADDED COMP.
009300    02 HI-KEY-ADDED   PIC 9(10).
009400    02 LOW-KEY-ADDED PIC 9(10).
009500 01 HIGHEST-LOWEST-KEYS REDEFINES KEYS-ADDED.
009600    02 HIGHEST-KEY-ADDED PIC X(5).
009700    02 LOWEST-KEY-ADDED  PIC X(5).
009800 01 ROUGH-TABLE.
009900    02 RT-KEY         PIC X(5) OC 1000.
009950*
010000*     THE # OF ROUGH-TABLE ENTRIES IS A COMPILE-TIME CONSTANT.
010100 01 RT-ENTRIES        PIC 9(6) COMP.
010120*
010150*     LAST-RT-KEY IS A REUSABLE WORK AREA IN ACTUAL CODE.
010200 01 LAST-RT-KEY       PIC X(5).
010300 01 LAST-RT-KEY-COMP REDEFINES LAST-RT-KEY PIC 9(10) COMP.
010320*
010350*     USER DATA STARTS HERE:
010370*
010400 01 FILE-STATUS       PIC 99.
010450*
010500 01 DATAFILE-ID.
010600    02 FILLER         PIC X(10) VA SPACES.
010700    02 D-FID          PIC X(10) PIC X(10).
010800    02 FILLER         PIC X(10) VA SPACES.
010900 01 TAGFILE-ID.
011000    02 FILLER         PIC X(10) VA SPACES.
011050    02 FILLER         PIC X(3)  VA "TAG".                 X$RPGTAGS
011100    02 T-FID          PIC X(7).
011200    02 FILLER         PIC X(10) VA SPACES.
011250********* NOTE: IF $RPGTAGS, "TAG" APPEARS IN FID INSTEAD OF MFID
011300 01 FOUND             PIC 9(4) COMP.
011400 01 NOT-FOUND         PIC 9(4) COMP.
011500 01 DISK-READS        PIC 9(8) COMP.
011600 01 SEED              PIC 9(18) COMP VA 557442103449672004.
011700 01 RANDOM-KEY REDEFINES SEED PIC 9(4) COMP.
```

```
011800 01 SQ                PIC 9(36) COMP.
011900 01 SEED-SQ REDEFINES SQ COMP.
012000   02 FILLER           PIC 9(18).
012100   02 MID              PIC 9(18).
012200   02 FILLER           PIC 9(18).
012250
012300 PROCEDURE DIVISION.
012400 DECLARATIVES.
012500 PARITY-ERROR SECTION.
012600     USE AFTER STANDARD ERROR PROCEDURE ON TAGFILE DATAFILE
012700       S-TAGFILE S-DATAFILE.
012800 P1.
012900     MOVE 30 TO FILE-STATUS.
013000 END DECLARATIVES.
013050
013100 STARTP.
013200$VOID 027500
027600 SORTER.
027700     SORT SD-TAGFILE
027800         ON ASCENDING KEY SD-KEY
027900         USING TAGFILE
028000         GIVING TAGFILE WITH LOCK.
028050
028100 OPEN-INPUT.
028200     OPEN INPUT TAGFILE.
028300     MOVE 1 TO OPEN-TYPE.
028400     MOVE 0 TO FILE-STATUS.
028600*
028700*    THE FOLLOWING MAY BE OMITTED FOR SEQUENTIAL FILES
028800*    UNLESS REFERENCED BY A START STATEMENT
028900*
029000     PERFORM BUILD-ROUGH-TABLE.
029100*
029200*    THE FOLLOWING IS NEEDED FOR SEQ FILES OR DYNAMIC WITH READ...
029300*
029400     MOVE 0 TO T-REC-NO.
029500*
029600     OPEN INPUT DATAFILE.
029650
029700 OPEN-I-O.
029750     OPEN I-O TAGFILE.
029800     MOVE 3 TO OPEN-TYPE.
029900     MOVE 0 TO FILE-STATUS.
029910*
029920*    THE FOLLOWING MAY BE OMITTED FOR SEQ FILES
029930*    UNLESS REFERENCED BY A START STATEMENT.
029940*
029950     PERFORM BUILD-ROUGH-TABLE.
030000*
030100*    THE FOLLOWING IS NEEDED FOR SEQ FILES OR DYNAMIC READ...NEXT
030200*
030300     MOVE 0 TO T-REC-NO.
030700     MOVE 0 TO PRIOR-KEY-COMP.
030800*
```

```
030900*     THE FOLLOWING IS NEEDED FOR SEQ FILES WITH REWRITE
031000*
031100      MOVE 0 TO LAST-I-O-WAS-A-READ-V.
031150*
031200*     THE FOLLOWING IS NEEDED FOR SEQ FILES WITH START
031250*
031300      MOVE ALL 9FF9 TO LOW-KEY-ADDED.
031350      MOVE 0 TO HI-KEY-ADDED.
031400*
031500      OPEN I-O DATAFILE.
031550
031600 OPEN-OUTPUT-SEQUENTIAL.
031650      OPEN OUTPUT S-TAGFILE.
031700      MOVE 2 TO OPEN-TYPE.
031800      MOVE 0 TO FILE-STATUS.
031900      MOVE 0 TO PRIOR-KEY-COMP.
032000      MOVE 0 TO S-T-INDEX.
032100      OPEN OUTPUT S-DATAFILE.
032200
032300 OPEN-OUTPUT.
032400      OPEN O-I TAGFILE.
032500      MOVE 2 TO OPEN-TYPE.
032600      MOVE 0 TO FILE-STATUS.
032650      MOVE 0 TO EOF-POINTER.
032700      MOVE 0 TO NEW-EOF-POINTER.
032800      MOVE ALL 9FF9 TO LOW-KEY-ADDED.
032900      MOVE 0 TO HI-KEY-ADDED.
033000      OPEN OUTPUT DATAFILE.
033050
033100 CLOSE-SEQUENTIAL-OUTPUT-RELEAS.
033200      CLOSE S-DATAFILE WITH RELEASE.
033300      CLOSE S-TAGFILE WITH RELEASE.
033400      MOVE 0 TO FILE-STATUS.
033450
033500 CLOSE-WITH-RELEASE.
033600      CLOSE DATAFILE WITH RELEASE.
033700      CLOSE TAGFILE WITH RELEASE.
033800      MOVE 0 TO FILE-STATUS.
033850*
033900*     THE FOLLOWING NEEDED IF WRITE USED AND NOT CLOSED PURGE.
033950*
034000      IF EOF-POINTER NOT = NEW-EOF-POINTER PERFORM SORTER.
034050*
034100*     THE FOLLOWING NEEDED FOR SIMULATION ONLY
034200*
034300      MOVE NEW-EOF-POINTER TO EOF-POINTER.
034400*
034500 READ-TAGFILE.
034600      READ TAGFILE.
034650
034700 BUILD-ROUGH-TABLE SECTION.
034800 GET-EOF-POINTER.
034900*
035000*     THE EOF-POINTER IS OBTAINED FROM AN
```

```
035100*     ACCESS.FILE.INFORMATION COMMUNICATE EXECUTED HERE
035200*
035250      IF EOF-POINTER = 0 GO TO REAL-XIT.
035300      MOVE EOF-POINTER TO NEW-EOF-POINTER.
035400      COMPUTE RT-INCREMENT = (EOF-POINTER + RT-ENTRIES - 1) /
035500        RT-ENTRIES.
035600*
035700*     CHECK FOR EOF-POINTER < # ROUGH TABLE ENTRIES
035800*
035900      IF RT-INCREMENT = 0
036000        MOVE 1 TO RT-INCREMENT.
036100      MOVE 0 TO LAST-RT-KEY-COMP.
036200      MOVE 0 TO RT-MAX.
036300      MOVE 0 TO T-REC-NO.
036400 LOOP.
036500      ADD RT-INCREMENT TO T-REC-NO.
036600      IF T-REC-NO > EOF-POINTER
036700        MOVE 0 TO T-REC-NO
036800        GO TO RT-EXIT.
036900      ADD 1 TO RT-MAX.
037000      PERFORM READ-TAGFILE.
037100 SEQUENCE-CHECK.
037200      IF T-KEY NOT > LAST-RT-KEY AND RT-MAX NOT = 1
037300        DISPLAY "TAGFILE SEQUENCE ERROR - DS OR DP"
037400        GO TO SEQUENCE-CHECK.
037500      MOVE T-KEY TO LAST-RT-KEY.
037600      MOVE T-KEY TO RT-KEY(RT-MAX).
037700      GO TO LOOP.
037800 RT-EXIT.
037900      MOVE EOF-POINTER TO T-REC-NO.
038000      PERFORM READ-TAGFILE.
038050      MOVE T-KEY TO EOF-KEY.
038100 REAL-XIT.
038150
038200 BINARY-SEARCH SECTION.
038300 B1.
038400      IF EOF-POINTER = 0
038500        MOVE 0 TO T-REC-COMP
038600        GO TO CHECK-ADDITIONS.
038700      MOVE 0 TO LO.
038800      MOVE RT-MAX TO HI.
038900 AGAIN.
039000      COMPUTE HALF = 5 * (HI + LO).
039100      IF LO = T-REC-NO
039200        GO TO SEARCH-DISK.
039300      IF ARGUMENT NOT > RT-KEY(T-REC-NO)
039400        MOVE T-REC-NO TO HI
039500        GO TO AGAIN.
039600      MOVE T-REC-NO TO LO.
039700      GO TO AGAIN.
039800 SEARCH-DISK.
039900      IF ARGUMENT = RT-KEY(HI)
040000        COMPUTE T-REC-NO = HI * RT-INCREMENT
040100        GO TO READ-THE-TAG-FILE.
```

```
040200      COMPUTE LO = LO * RT-INCREMENT.
040300      COMPUTE HI = HI * RT-INCREMENT.
040400      IF HI NOT > EOF-POINTER
040500        GO TO SEARCH-DISK-LOOP.
040600      IF ARGUMENT > EOF-KEY
040700        GO TO CHECK-ADDITIONS.
040800      ADD 1 EOF-POINTER GIVING HI.
040900 SEARCH-DISK-LOOP.
041000      IF LO + 1 NOT < HI
041100        GO TO CHECK-ADDITIONS.
041200      COMPUTE HALF = 5 * (LO + HI).
041300 READ-THE-TAG-FILE.
041400      IF T-REC-NO = 0
041500        GO TO NOT-FOUND.
041600      PERFORM READ-TAGFILE.
041700*
041800*     THE FOLLOWING IS NEEDED FOR DEBUGGING
041900*
042000      ADD 1 TO DISK-READS.
042100*
042200      IF ARGUMENT = T-KEY
042300        GO TO FOUND.
042400      IF ARGUMENT NOT > T-KEY
042500        MOVE T-REC-NO TO HI
042600        GO TO SEARCH-DISK-LOOP.
042700      MOVE T-REC-NO TO LO.
042800      GO TO SEARCH-DISK-LOOP.
042900 CHECK-ADDITIONS.
043000      IF ARGUMENT > HIGHEST-KEY-ADDED
043100        GO TO NOT-FOUND.
043200      IF ARGUMENT < LOWEST-KEY-ADDED
043300        GO TO NOT-FOUND.
043400      MOVE EOF-POINTER TO T-REC-NO.
043500 SEARCH-ADDITIONS.
043600      ADD 1 TO T-REC-NO.
043700      IF T-REC-NO > NEW-EOF-POINTER
043800        GO TO NOT-FOUND.
043900      PERFORM READ-TAGFILE.
044000*
044100*     THE FOLLOWING IS NEEDED FOR DEBUGGING
044200*
044300      ADD 1 TO DISK-READS.
044400*
044500      IF T-KEY NOT = ARGUMENT
044600        GO TO SEARCH-ADDITIONS.
044700 FOUND.
044800*
044900*     THE FOLLOWING NEEDED FOR DEBUGGING
045000*
045100      ADD 1 TO FOUND.
045200*
045300      GO TO SEARCH-EXIT.
045400 NOT-FOUND.
045500*
```

```
045600*     THE FOLLOWING NEEDED FOR DEBUGGING
045700*
045800      ADD 1 TO NOT-FOUND.
045900*
046000      MOVE 0 TO T-REC-NO.
046100 SEARCH-EXIT.
046150
046200 SORT-THE-TAGFILE SECTION.
046300 S1.
046400      CLOSE TAGFILE LOCK.
046500      PERFORM SORTER.
046510*
046520*   THE FOLLOWING IS NEEDED FOR SIMULATION ONLY.
046530*
046540      MOVE NEW-EOF-POINTER TO EOF-POINTER.
046550*
046600      OPEN I-O TAGFILE.
046650      PERFORM BUILD-ROUGH-TABLE.
046700
046900 READ-SEQUENTIAL SECTION.
047000 R1.
047050      MOVE 0 TO FILE-STATUS.
047100*
047200*     THE FOLLOWING MAY BE OMITTED FOR SEQ FILES OR NO WRITES.
047300*
047400      IF EOF-POINTER = NEW-EOF-POINTER GO TO SETUP.
047410      MOVE SAVE-T-REC-NO TO T-REC-NO.
047420      READ TAGFILE.
047430      MOVE T-KEY TO ARGUMENT.
047450      PERFORM SORT-THE-TAGFILE
047500      PERFORM BINARY-SEARCH.
047600*
047700 SETUP.
047900      ADD 1 TO T-REC-NO.
048000      READ TAGFILE AT END
048100        MOVE 10 TO FILE-STATUS
048200        SUBTRACT 1 FROM T-REC-NO
048300        GO TO R1-EXIT.
048350*
048400*     THE FOLLOWING MOVE NECESSARY ONLY IN MODEL -
048450*         SAME FIELD IN ACTUAL CODE
048500*****************
048550      MOVE T-INDEX TO D-REC-NO.
048600*****************
048700*     THE FOLLOWING IS NEEDED FOR SEQUENTIAL FILES
048750*     FOR WHICH A REWRITE IS SPECIFIED
048800*
048850      MOVE 1 TO LAST-I-O-WAS-A-READ-V.
048900*
049000      READ DATAFILE.
049050      IF T-KEY NOT = D-KEY
049100          MOVE 93 TO FILE-STATUS
049120*             NOTE: SEE ALSO READ-RANDOM IF FILE-STATUS NOT PRESENT
049150      ELSE
```

```
049200          MOVE D-KEY TO PRIOR-KEY.
049250 R1-EXIT.
049300
049400 READ-RANDOM SECTION.
049500 R2.
049600      MOVE 0 TO FILE-STATUS.
049700      IF OPEN-TYPE = 0
049750        STOP "READ ON UNOPENED FILE - DS OR DP".
049800      MOVE D-KEY TO ARGUMENT.
049850      PERFORM BINARY-SEARCH.
049900      IF T-REC-NO = 0
050000        MOVE 23 TO FILE-STATUS
050100        GO TO R2-EXIT.
050150*****************
050200      MOVE T-INDEX TO D-REC-NO.
050250******************
050300      READ DATAFILE.
050400
050500      IF T-KEY NOT = D-KEY
050600          MOVE 93 TO FILE-STATUS.
050700*             IF FILE-STATUS NOT PRESENT DO THIS INSTEAD:
050800*          STOP "TAG FILE NOT SYNCHRONIZED WITH DATA FILE - DS OR DP
050900 R2-EXIT.
050950
051000 WRITE-SEQUENTIAL SECTION.
051100 W1.
051150      MOVE 0 TO FILE-STATUS.
051200      IF OPENED-I-O
051300        STOP "WRITE ON A SEQUENTIAL I-O FILE - DS OR DP"
051400        GO TO W1.
051500      IF S-D-KEY NOT > PRIOR-KEY AND S-T-INDEX NOT = 0
051600        MOVE 21 TO FILE-STATUS
051700        GO TO W-SEQ-EXIT.
051800      MOVE S-D-KEY TO PRIOR-KEY.
051900      MOVE S-D-KEY TO S-T-KEY.
052000 W2.
052100      ADD 1 TO S-T-INDEX ON SIZE ERROR
052200        STOP "TAGFILE SIZE EXCEEDED - DS OR DP"
052300        GO TO W2.
052400      WRITE S-T-REC.
052500      WRITE S-D-REC.
052600 W-SEQ-EXIT.
052650
052700 WRITE-RANDOM SECTION.
052750 W1.
052800      MOVE 0 TO FILE-STATUS.
052850      IF OPEN-TYPE = 0
052900        STOP "WRITE ON UNOPENED FILE - DS OR DP".
052950      MOVE D-KEY TO ARGUMENT.
052970      MOVE T-REC-NO TO SAVE-T-REC-NO.
053000      PERFORM BINARY-SEARCH.
053100      IF T-REC-NO NOT = 0
053200        MOVE 22 TO FILE-STATUS
053300        GO TO W-RANDOM-EXIT.
```

```
053400      MOVE D-KEY TO T-KEY.
053500 W2.
053600      ADD 1 TO NEW-EOF-POINTER ON SIZE ERROR
053700        STOP "TAGFILE SIZE EXCEEDED - DS-OR DP"
053800        GO TO W2.
053900      MOVE NEW-EOF-POINTER TO T-REC-NO, T-INDEX.
054000
054100      WRITE T-REC.
054150*****************
054200      MOVE T-INDEX TO D-REC-NO.
054250*****************
054400      IF ARGUMENT < LOWEST-KEY-ADDED
054500        MOVE ARGUMENT TO LOWEST-KEY-ADDED.
054600      IF ARGUMENT > HIGHEST-KEY-ADDED
054700        MOVE ARGUMENT TO HIGHEST-KEY-ADDED.
054750      WRITE D-REC.
054800      MOVE SAVE-T-REC-NO TO T-REC-NO.
054850 W-RANDOM-EXIT.
054900
054950 REWRITE-SEQUENTIAL SECTION.
055000 RE-1.
055100      IF NOT OPENED-I-O
055200        STOP "REWRITE ON A FILE NOT OPENED I-O - DS OR DP"
055300        GO TO RE-1.
055400      MOVE 0 TO FILE-STATUS.
055800      IF NOT (LAST-I-O-WAS-A-READ AND D-KEY = PRIOR-KEY)
055900        MOVE 21 TO FILE-STATUS
056000        GO TO REWRITE-SEQ-EXIT.
056600      MOVE 0 TO LAST-I-O-WAS-A-READ-V.
056700      WRITE D-REC.
056800
056850 REWRITE-SEQ-EXIT.
056860
056900 REWRITE-RANDOM SECTION.
057000 RE-1.
057100      IF NOT OPENED-I-O
057200        STOP "REWRITE ON A FILE NOT OPENED I-O - DS OR DP"
057300        GO TO RE-1.
057400      MOVE 0 TO FILE-STATUS.
057500      MOVE D-KEY TO ARGUMENT.
057550      MOVE T-REC-NO TO SAVE-T-REC-NO.
057600      PERFORM BINARY-SEARCH.
057700      IF T-REC-NO = 0
057800        MOVE 23 TO FILE-STATUS
057900        GO TO REWRITE-RANDOM-EXIT.
057950*****************
058000      MOVE T-INDEX TO D-REC-NO.
058050*****************
058100      WRITE D-REC.
058150      MOVE SAVE-T-REC-NO TO T-REC-NO.
058200 REWRITE-RANDOM-EXIT.
058250
058300 START-SETUP SECTION.
058400 S1.
```

```
058500      IF OPENED-OUTPUT OR OPEN-TYPE = 0
058600        STOP "START ON FILE NOT OPENED INPUT OR I-O -DS OR DP"
058700        GO TO S1.
058800*
058900*     THE FOLLOWING MAY BE OMITTED FOR SEQ FILES OR NO WRITES.
059000*
059100      IF EOF-POINTER NOT = NEW-EOF-POINTER
059200        PERFORM SORT-THE-TAGFILE.
059300*
059400      MOVE 0 TO FILE-STATUS.
059500*
059600*     THE FOLLOWING IS NEEDED FOR SEQUENTIAL FILES
059700*     FOR WHICH A REWRITE IS SPECIFIED
059800*
059900      MOVE 0 TO LAST-I-O-WAS-A-READ-V.
060000*
060100      MOVE 0 TO KEY TO ARGUMENT.
060150      PERFORM BINARY-SEARCH.
060200
060300 START-EQL SECTION.
060400 S1.
060500      PERFORM START-SETUP.
060600      IF T-REC-NO = 0
060700        MOVE 23 TO FILE-STATUS
060800        MOVE EOF-POINTER TO T-REC-NO
060900        GO TO START-EQL-EXIT.
061000      SUBTRACT 1 FROM T-REC-NO.
061100 START-EQL-EXIT.
061150
061200 START-NOT-LSS SECTION.
061300 S1.
061400      PERFORM START-SETUP.
061500      IF T-REC-NO NOT = 0
061600        GO TO ITS-EQL.
061650      IF ARGUMENT > EOF-KEY GO TO INV-KEY.
061700      COMPUTE HALF = 5 * (LO + HI).
061800      IF T-REC-NO < EOF-POINTER
061900        GO TO START-NOT-LSS-EXIT.
062000 INV-KEY.
062100      MOVE 23 TO FILE-STATUS.
062200      MOVE EOF-POINTER TO T-REC-NO.
062300      GO TO START-NOT-LSS-EXIT.
062400 ITS-EQL.
062500      SUBTRACT 1 FROM T-REC-NO.
062600 START-NOT-LSS-EXIT.
062650
062700 START-GTR SECTION.
062800 S1.
062900      PERFORM START-SETUP.
063000      IF T-REC-NO NOT = 0
063050        GO TO ITS-EQL-2.
063100      IF ARGUMENT > EOF-KEY GO TO INV-KEY-2.
063150      COMPUTE HALF = 5 * (HI + LO).
063200      IF T-REC-NO < EOF-POINTER
```

```
063300         GO TO START-GTR-EXIT.
063400 INV-KEY-2.
063500       MOVE 23 TO FILE-STATUS.
063600       MOVE EOF-POINTER TO T-REC-NO.
063700       GO TO START-GTR-EXIT.
063800 ITS-EQL-2.
064000       IF T-REC-NO NOT < EOF-POINTER
064100         GO TO INV-KEY-2.
064500 START-GTR-EXIT.
```

BURROUGHS CORPORATION
COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

DATA MANAGEMENT

This portion of the specification documents the Data Management section of the COBOL compiler and is subdivided into sections which concern the declaration, processing, and accessing of DM entries. A sample program is included to support the discussion of data management.

DATA-BASE DECLARATIONS

DB [<logical data-base name> OF] <data-base name>.

DATASET

<level-number> <dataset-name> [ORDERED/RESTART] DATA SET (<DDL #>).
[<set-name> [RETRIEVAL] SET (<DDL #>, AUTO)
OF <dataset-name> (<DDL #>)
KEY IS/KEYS ARE <component-name-1>[,<component-name-2>]... .

SUBSET

<level-number> <subset-name> SET (<DDL #>, MANUAL)
OF <dataset-name> (<DDL #>)
[KEY IS/KEYS ARE <component-name-1>[,<component-name-2>]...].

FORMAT OF DDL-NUMBER

The <DDL #> consists of a structure number followed by the time and date when the dataset was created. The format for DDL-NUMBER is:

SSS HH:MM:SS MM/DD/YY

This string of characters is encoded into a 64-bit path dictionary entry by MERGE.

```
01 PATH.DICT.FORMAT          BIT(64),
  02 LOCK.BIT                BIT(1),
  02 MEDIA.BIT               BIT(1),
  02 BEEN.OPENED             BIT(1),
  02 EXPLICIT.OPEN           BIT(1),
```

```
02 FILLER                    BIT(4),
02 STR.NUMBER                BIT(8),
02 DM.TIME                   BIT(17),
  03 DM.HOUR                 BIT(5),
  03 DM.MIN                  BIT(6),
  03 DM.SEC                  BIT(6),
02 DM.DATE                   BIT(16),
  03 DM.MONTH                BIT(4),
  03 DM.DAY                  BIT(5),
  03 DM.YEAR                 BIT(7),
```

DMSTATUS REGISTER

The DMSTATUS REGISTER is six digits long and will be set by each DM communicate. A value of zero indicates no exception.

PARSE — Adds DMSTATUS to the reserved word lists and translates the syntax of DMSTATUS (<exception name>) into (DMSTATUS = <literal>). <exception-name>s are reserved words only in this context. Each corresponds to an exception-number <literal>.

DMCATEGORY becomes a separate TRESWD temporarily.

MERGE — Assigns memory in DSEGO.

EXPLODE — Expands references to look like a TWORD, and sets CONDITIONS.NAME flag so DMSTATUS will always be a full relation in PROSYN. This also prevents DMSTATUS from being used outside a condition.

For DMCATEGORY, transform to a normal TWORD, allowing direct access by any appropriate COBOL operation.

PARSE

When the DATA-BASE SECTION is declared, the compiler does the following:

Verifies that it follows the FILE SECTION and precedes the WORKING-STORAGE SECTION.

Outputs a dummy TXON with level = 0 (like WS-SECTION) and symbol = "DATA-BASE" followed by the DB.PACK.ID obtained from the FPB of the library file.

Saves this DB.SECTION.OCUR as a global.

DATABASE.SECTION.FLAG := 1

Verifies that the DATA-BASE SECTION consists only of "01 DS-NAME INVOKE ..." entries and "DB ..." entries.

When DB is declared, the compiler:

Saves the data-base name.

Outputs the data-base name as a TNNLIT.

Outputs a TXDN.

If present, outputs "OF" followed by a data-base name as a TNNLIT, which effectively redefines the previous data-base name as <logical data-base name>.

When INVOKE is declared, the compiler:

Verifies DATABASE.SECTION.FLAG = 1 and LEVEL = 01.

Opens #DATA-BASE/DATASET.NAME (like a copy from library) where data-base is the first nine characters of the data-base name.

Scans past the 01 dataset-name of the source image and substitutes the invoking dataset-name.

Marks the scanned source images with "*".

When DATA SET is declared, the compiler:

Verifies DATA.BASE.SECTION.FLAG = 1

Parses the DDL-NUMBER.

Assigns an OCUR to each key-name where level := IF "RETRIEVAL" THEN 52 ELSE 51

Marks component-names as TWORD with "DATA-BASE" as a TQUAL.

Checks for "RESTART" or "ORDERED" and sends it thru to Merge.

When SET is declared, the compiler:

Verifies DATABASE.SECTION.FLAG = 1

Parses the DDL-NUMBER

Level : 53

Deletes the target dataset-name

Parses the target DDL-NUMBER

Outputs the target component-name as TWORD with "DATA-BASE" as TQUAL.

When end of DATA-BASE SECTION is found, DATABASE.SECTION.FLAG := 0.

When FIND/MODIFY is specified, the compiler outputs the component-names of the selection expression with "DATA-BASE" as TQUAL.

DICTIONARY PROCESSING

No changes necessary for Data Management.

QUALIFICATION RESOLUTION

The following characteristics are exhibited:

Items with levels of 52, 52, or 53 have scope only of themselves.

The DB.SECTION.OCUR includes all database items in its scope.

Inhibits error reporting for a TWORD with the DB.SECTION.OCUR as its qualifier. If unidentified name, then OCUR := 0; if insufficient qualification, then OCUR := last of the duplicates.

Note: No condition-names will be contained in a DATASET description, therefore the building of the CONDITION.NAME.TABLE does not conflict with the building of the PATH.INFO.TABLE.

MERGE

DATA-BASE SECTION PROCESSING

Save the DB.PACK.ID (TNNLIT) to be used in DM.OPEN and DM.CLOSE.

For each 01 dataset-name, INVOKE.NO := BUMP INVOKE.COUNT

Based on the STRUCTURE.NUMBER of the DDL-NUMBER, the compiler assigns a unique PATH.NO and builds a corresponding PATH.DICTIONARY.ENTRY in the PATH.DICTIONARY block on the SEGFILE which contains the unique DDL-NUMBER (Start the PATH.NO at one).

Build a table relating the dataset OCUR to its key-names and their component-names and containing the INVOKE.NO and PATH.NO and the TARGET.PATH.NO. Note: If the target DATASET of a SUBSET is not INVOKED, then ignore that path and its components when building the table, e.g., its component OCUR=0. The "RESTART" attribute is also encoded in this table entry.

Drop all attributes (e.g., DDL-NUMBER, component-names, etc.) when parsing path declarations.

For each DB, save the data-base name (TNNLIT) in a table with its OCUR. Drop TNNLIT. Logical-data-base name (another TNNLIT followed by "OF" if present) will replace data-base name in this table, for eventual use in "OPEN" communicate.

Example 1:

```
02 STUDENTS ORDERED DATA SET (38).
   STUDNAMES SET (71, AUTO) OF STUDENTS (38)
   KEYS ARE LAST-NAME, FIRST-NAME.
```

ALL	DN	ADN
TXDN	TXDN (STUDENTS,LEVEL=02)	TXDN (STUDENTS,LEVEL=02)
ORDERED		
DATA		DATA
TINTGR(38)		
TXDN	TXDN (STUDENTS,LEVEL=51)	TXDN (STUDNAMES,LEVEL=51)
TINTGR(71)		
TWORD	TWORD(LAST-NAME)	
TWORD	TWORD(FIRST-NAME)	

Example 2:

```
02 COURSE SET (8,MANUAL)
   OF UNIV-COURSES (14)
   KEYS ARE DEPARTMENT,LEVEL.
```

ALL	ON	ADN
TXDN TINTGR(8) OF TINTGR(14)	TXDN (COURSE,LEVEL=53)	TXDN (COURSE,LEVEL=53)
TWORD	TWORD(DEPARTMENT)	
TWORD	TWORD(LEVEL)	

PROCEDURE DIVISION PROCESSING

GENERAL

Parse the DM.VERBS with particular attention to the relationship of dataset-names and their paths.

Output the INVOKE.NO and PATH.NO as TINTGR tokens following references to patch names.

The CT.ADVERB bits for the DM.VERBS are arranged as follows:

BIT	MEANING

1 Distinguishes INSERT from STORE or REMOVE from DELETE.
2 RECREATE or distinguishes BEGIN-TRANSACTION from
END-TRANSACTION
3 DMSTATUS format:
0:Binary
1:Decimal (4 BIT)
4 ON EXCEPTION specified
5 UPDATE
6 MODIFY
7-11 SELECTION EXPRESSION type
0:NEXT
1:PRIOR
2:FIRST
3:LAST
4:NEXT and "AT" clause
5:CURRENT
6:"AT" clause

12 PATH TYPE
0:KEY
1:DATASET or

Note: = No syntax checking will be done by MERGE for the EXCEPTION clause. If the clause is specified, it will be output as is, for handling by PROSYN.

OPEN

Defined: OPEN UPDATE/INQUIRY <data-base name>
[ON EXCEPTION <statement> [ELSE <statement>]]

The communicates used in OPEN are as follows:

CT.VERB=6
CT.OBJECT=Not used
CT.ADVERB=
BIT 1 = PACK.ID included
BIT 3 = DMSTATUS format
0:Binary
1:Decimal (4 BIT)
BIT 4 = "ON EXCEPTION"
BIT 5 = "UPDATE"
0: "INQUIRY"
1: "UPDATE"
CT.1=Bit length of DMSTATUS register

CT.2=Base relative address of DMSTATUS register
CT.3=Base relative address of data-base name
CT.4=Length of data-base name in bits
CT.5=Base relative address of data-base PACK.ID
CT.6=Length of data-base PACK.ID

MERGE OUTPUT

Defined: TTRESWD OPEN.DM
TNNLIT data-base name
TRESWD UPDATE <exception clause>

Verify that it is a data-base name

Change OPEN to DM.OPEN before output (new verb)

Look up data-base name in table by OCUR. output the name.

CLOSE

Defined = CLOSE <data-base name>
[ON EXCEPTION <statement> ELSE <statement>]

CT.VERB=7
CT.OBJECT=Not used
CT.ADVERB=
BIT 1 = PACK.ID included
BIT 3 = DMSTATUS format
0:Binary
1:Decimal (4 bit)
BIT 4 = "ON EXCEPTION"
CT.1=Bit length of DMSTATUS register
CT.2=Base relative address of DMSTATUS register
CT.3=Base relative address of data-base name
CT.4=Length of data-base name in bits
CT.5=Base relative address of data-base PACK.ID
CT.6=Length of data-base PACK.ID

MERGE OUTPUT

Defined: TRESWD CLOSE.DM
TNNLIT data-base name
<EXCEPTION CLAUSE>

Verify that it is a data-base name

Change CLOSE to DM.CLOSE before output (new verb)

Look up data-base name in table by OCUR, output the name.

CREATE/RECREATE

Defined: CREATE/RECREATE <dataset-name>
[ON EXCEPTION <statement>[ELSE <statement>]]

CT.VERB=18
CT.OBJECT=INVOKE.NO, PATH.NO
CT.ADVERB=
BIT 3 = DMSTATUS format
0:Binary
1:Decimal (4 BIT)
BIT 4 = "ON EXCEPTION"
BIT 8 = "RECREATE"
CT.1=Bit length of DMSTATUS register
CT.2=Base relative address of DMSTATUS register
CT.3=Bit length of the DATASET record work area
CT.4=Base relative address of the DATASET record work area

MERGE OUTPUT

Defined: TRESWRD CREATE/RECREATE
TINTGR INVOKE.NUM of dataset-name
TINTGR PATH.NUM of dataset-name
TWORD dataset-name <exception clause>

Verify the dataset-name

Output the dataset-name and path information

STORE

Defined: STORE <dataset-name>
[ON EXCEPTION <statement> [ELSE <statement>]]

CT.VERB=16
CT.OBJECT=INVOKE.NO, PATH.NO
CT.ADVERB=
BIT 3 = DMSTATUS format
0:Binary
1:Decimal (4 BIT)
CT.1=Bit length of DMSTATUS register
CT.2=Base relative address of DMSTATUS register
CT.3=Bit length of the DATASET record work area
CT.4=Base Relative address of the DATASET record work area

MERGE OUTPUT

Defined:

TRESWD	STORE
TINTGR	INVOKE.NUM of dataset-name
TINTGR	PATH.NUM of dataset-name
TWORD	dataset-name <exception-clause>

Verify the dataset-name

Output the dataset-name and path information

FREE

Defined: FREE <dataset-name>
[ON EXCEPTION <statement> [ELSE <statement>]]

CT.VERB=21
CT.OBJECT=INVOKE.NO, PATH.NO
CT.ADVERB=
BIT 3 = DMSTATUS format
0:Binary
BIT 4 = "ON EXCEPTION"
CT.1=Bit length of DMSTATUS register
CT.2=Base relative address of DMSTATUS register

MERGE OUTPUT

```
TRESWD  FREE
TINTGR  INVOKE.NUM of dataset-name
TINTGTR  PATH.NUM of dataset-name
TWORD  dataset-name <exception clause>
```

Verify the dataset-name

Output the dataset-name and path information

INSERT/REMOVE

Defined: INSERT <dataset-name> INTO <subset-name>
[ON EXCEPTION <statement> [ELSE <statement>]]

Defined: REMOVE CURRENT FROM <subset-name>
[ON EXCEPTION <statement> [ELSE <statement>]]

```
CT.VERB=16 INVOKE.NO, PATH.NO of subset-name
CT.ADVERB=
  BIT 1 = ON
  BIT 3 = DMSTATUS format
    0:Binary
    1:Decimal (4 bit)
  BIT 4 = "ON EXCEPTION"
CT.1=Bit length of DMSTATUS register
CT.2=Base relative address of DMSTATUS register
CT.3=INVOKE.NO, PATH.NO of dataset-name
```

MERGE OUTPUT

```
TRESWD  INSERT/REMOVE
TINTGR  INVOKE.NUM of dataset-name
TINTGR  PATH.NUM of dataset-name
TWORD   dataset-name or TRESWD CURRENT
TINTGR  INVOKE.NUM of subset-name
TINTGR  PATH.NUM of subset-name
TWORD   subset-name  <exception clause>
```

Verify the dataset-name to be a proper target DATASET of the subset-name and output it with its path info.

Output the target dataset-name and its path information.

CT.VERBs are like those for STORE and DELETE. They are distinguished by BIT 1 on CT.VERB.

For REMOVE, INVOKE.NUM and PATH.NUM of DATASET are zero.

FIND/MODIFY

CURRENT PATH INFORMATION

Defined: FIND/MODIFY [<dataset-name> VIA]
<dataset-name/<subset-name>/<order-name>/
<retrieval-name>
[ON EXCEPTION <statement> [ELSE <statement>]]

ORDERED ACCESS FORMAT

Defined: FIND/MODIFY [<dataset-name> VIA]
NEXT/PRIOR/FIRST/LAST <dataset-name>/<subset-name>
<order-name>
[ON EXCEPTION <statement> [ELSE <statement>]]

RANDOM ACCESS FORMAT

Defined: FIND/MODIFY [dataset-name VIA] [NEXT]
<subset-name>/<order-name>/<retrieval-name> AT
<component-name-1> = <data-name-1>/<literal-1>
[AND <component-name-2> = <data-name-2>/
<literal-2>] . . .
[<ON EXCEPTION <statement> [ELSE <statement>]]

Note: The "<dataset-name> VIA" clause is mandatory when the path is a <subset-name>.

```
CT.VERB=15
CT.OBJECT=INVOKE.NO, PATH.NO of the path-name
CT.ADVERB=
  BIT 3 = DMSTATUS format
    0:Binary
    1:Decimal (4 BIT)
  BIT 4 = "ON EXCEPTION"
  BIT 6  = "MODIFY"
BIT 7-11 = Type of selection expression
    0:NEXT
    1:PRIOR
    2:FIRST
    3:LAST
```

```
        4:NEXT and AT clause
        5:Current
        6:AT clause
      BIT 12 = Type of path
        0:Key
        1:Dataset
```

CT.1=Bit length of DMSTATUS register
CT.2=Base relative address of DMSTATUS register
CT.3=Bit length of the DATASET record work area
CT.5=Base relative address of SEARCH KEY (CAT of component names)
CT.6=INVOKE.NO, PATH.NO of dataset-name

MERGE OUTPUT

```
TRESWD  FIND/MODIFY
TINTGR  INVOKE.NUM of dataset-name
TINTGR  PATH.NUM of dataset-name
TWORD   dataset-name
TINTGR  INVOKE.NUM of the path
TWORD   PATH.NAME
[TRESWD  NEXT/FIRST/PRIOR/LAST]
  <selection expression>
  <exception clause>
```

Where <SELECTION EXPRESSION> is:

```
TWORD   COMPONENT-NAME
TRESWD  "="
TWORD/LITERAL
    .
    .
    .
```

Verify the PATH.NAME to be a proper path to the dataset-name and output the path information.

Output the target dataset-name and its path information (supply the parent OCUR as a TWORD token if the dataset-name is omitted).

Verify the component-name list to be in the proper order with no names missing from the list.

The type of the PATH.NAME is DATASET if BIT 7 of the DNFLAGS is on in the PATH.NAME. This bit is used as the FILLER.FLAG in the DATA DIVISION.

DAISYN

When DATA SET is declared, allocate a separate work area for it.

When an entry with level=51, 52 or 53 is declared, it is treated like an 88, i.e., storage is not allocated.

Verify the uniqueness of the 01 DATASET-NAMES within a DB and DATA-BASE names.

Special handling needed for nested data sets to avoid scope-related control problems.

EXPLODE

Change TWORD to TINDEX for items with level=51-53 (done in PROCEDURE RESOLVE.FACTORS) in order to restrict the use of PATH.NAMES to DM.VERBS.

PROSYN PROCESSING

Parse the "ON EXCEPTION . . ." clause of the DM.VERBS.

CODEGEN

Generate communicates corresponding to DM verbs.

FIXUP PROCESSING

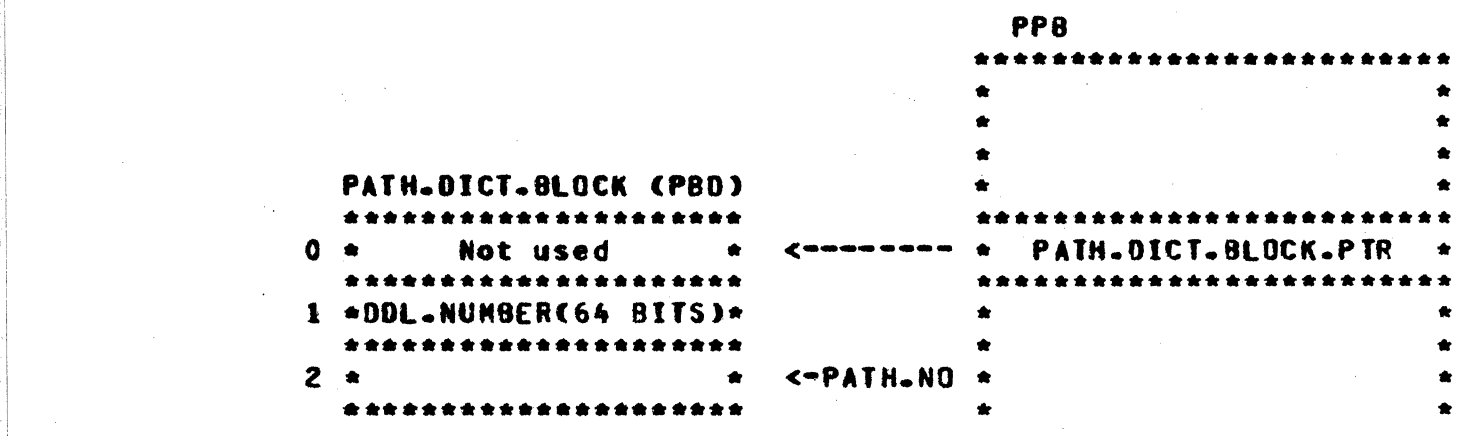


Table 3.1 FIXUP Processing

- The compiler assigns path numbers to each unique path name declared in the DATA-BASE SECTION (e.g., dataset-name, subset-name, order-name, retrieval-name).
- The PATH DICTIONARY BLOCK (PDB) contains a PATH DICTIONARY ENTRY (PDE) for each path name. The PDE contains the 64-bit DDL-NUMBER associated with that path name.
- The PATH.NO and INVOKE.NO are used to identify the desired path to a DATASET record. The PATH.NO is used as an index into the PATH.DICTIONARY to get the DDL-NUMBER. The INVOKE.NO is used to identify the information about a DATASET record where that DATASET has been invoked more than once.

SAMPLE DATA-BASE SECTION

```
                                          OCUR INV# PATH# TGT# CN CN
                                          ---- ---- ----- ---- -- --

DATA-BASE SECTION.                         01
DB UNIVERSITIES.                           02
01 STUDENT INVOKE UNIV-PERSONNEL.

 *01 UNIV-PERSONNEL DATA SET (88).          03   01    01    01
 *   PERSNAMES SET (73, AUTO)              04   01    02    01   25 27
 *     OF UNIV-PERSONNEL (88)
 *     KEYS ARE LASTNAME,FIRSTNAME.
 *   PERSMAJORS RETRIEVAL SET (82,AUTO)    05   01    03    01   34
 *     OF UNIV-PERSONNEL (88)
 *     KEY IS MAJOR.
 *   PERSAGES RETRIEVAL SET (2, AUTO)      06   01    04    01   29
```

```
 *      OF UNIV-PERSONNEL (88)                          
 *      KEY IS AGE.                                     
 *    02 NAME.                                  07      
 *      03  LASTNAME PC X(15).                  08      
 *      03  INITIAL PC X.                       09      
 *      03  FIRSTNAME PC X(10).                 10      
 *    02  SEX PC 9 CMP.                         11      
 *    02  AGE PC 99 CMP.                        12      
 *    02  CORSE-NOS OCCURS 4.                   13      
 *      03 DEPT PC XX.                           14      
 *      03  NMBR PC 999 CMP.                    15      
 *    02  COURSE SET (8, MANUAL)                16   01   05   06   41 42
 *        OF UNIV-COURSES (14)                          
 *        KEYS ARE DEPARTMENT, LEVEL.                   
 *    02  MAJOR  PC X(4)                        17      
 *    02  SALARY  PC 9(5)V99 CMP.              18      
 *    02  ADVISOR SET (16, MANUAL)              19   01   07   01      
 *        OF UNIV-PERSONNEL (88).                       
01 FACULTY INVOKE UNIV-PERSONNEL.                       
 *01 UNIV-PERSONNEL DATA SET (88).              20   02   01   01      
 *    PERSNAMES SET (73, AUTO)                  21   02   02   01   25 27
 *      OF  UNIV-PERSONNEL (88)                       
 *      KEYS ARE LASTNAME, FIRSTNAME.                 
 *    PERSMAJORS RETRIEVAL SET (82, AUTO)       22   02   03   01   34      
 *      OF UNIV-PERSONNEL (88)                        
 *      KEY IS MAJOR.                                 
 *    PERSAGES RETRIEVAL SET (2, AUTO)          23   02   04   01   29      
 *      OF UNIV-PERSONNEL (88)                        
 *      KEY IS AGE.                                   
 *    02  NAME.                                 24      
 *      03  LASTNAME  PC X(15).                 25      
 *      03  INITIAL PC X.                       26      
 *      03  FIRSTNAME PC X(10).                 27      
 *    02  SEX PC 9 CMP.                         28      
 *    02  AGE PC 99 CMP.                        29      
 *    02  CORSE-NOS OCCURS 4.                   30      
 *      03  DEPT PC XX.                         31      
 *      03  NMBR  PC  999 CMP.                  32      
 *    02 COURSE SET (8, MANUAL)                 33   02   05   06   41   42
 *       OF UNIV-COURSES (14)                         
 *       KEYS ARE DEPARTMENT, LEVEL.                  
 *    02  MAJOR  PC X(4).                       34      
 *    02  SALARY  PC 9(5)V99  CMP.              35      
 *    02  ADVISOR SET (16, MANUAL)              36   02   07   01      
 *          OF UNIV-PERSONNEL (88).                     
01  U-COURSES INVOKE UNIV-COURSES.                     
 *01  UNIV.COURSES DATA SET (14).               37   03   06   06      
 *    DEPTLEVELS SET (32, AUTO)                 38   03   08   06   41   42
 *      OF UNIV-COURSES (14)                          
 *      KEYS ARE DEPARTMENT, LEVEL.                   
 *    CLASS-SZ RETRIEVAL SET (34, AUTO)         39   03   09   06   50      
 *      OF UNIV-COURSES (14)                          
 *      KEY IS CLASS-SIZE.                            
 *    DEPTS RETRIEVAL SET (12, AUTO)            40   03   10   06   41      
```

```
*       OF UNIV-COURSES (14)
*       KEY IS DEPARTMENT.
*  02   DEPARTMENT  PC XX.                    41
*  02   LEVEL  PC 999 CMP.                    42
*  02   PROFESSOR SET (33, MANUAL)            43    03    11    01
*       OF UNIV-PERSONNEL (88).
*  02   BOOKS DATA SET (7).                   44    03    12    12
*    03 TITLE  PC  X(60).                     45
*    03 AUTHOR  PC X(30).                     46
*  02 DAYS-OF-WEEK  PC XXX.                   47
*  02 BUILDING  PC 999 CMP.                   48
*  02 ROOM PC XX.                             49
*  02 CLASS-SIZE  PC 99 CMP.                  50
*  02 STUDENTS ORDERED DATA SET (38).         51    03    13    13
*     STUDNAMES SET (71, AUTO)                52    03    14    13    53    54
*       OF STUDENTS (38)
*       KEYS ARE LAST-NAME, FIRST-NAME.
*    03 LAST-NAME  PC X(15).                  53
*    03 FIRST-NAME  PC X(10).                 54
```

PATH DICTIONARY TABLE

PATH.NO	STRUCTURE.NO
0	NOT USED
1	88
2	73
3	82
4	02
5	08
6	14
7	16
8	32
9	34
10	12
11	33
12	07
13	38
14	71

NOTES:

-- The source images with an "*" are provided by the DASDL Compiler as a library file to the COBOL Compiler.

-- The invoked name is the name of that file and is not a proper data-name of the DATASET (e.g., "#UNIVERSIT"/"UNIV-PERSO").

-- The name "STUDENT" replaces the word "UNIV-PERSONNEL" at OCUR=3.

-- The name "UNIV-PERSONNEL" is deleted when it appears as the target dataset-name of ADVISOR at OCUR=19 and OCUR=36.

-- The DATA-BASE SECTION appears to be a file name (OCUR=1) whose scope includes all DATA.BASE data-names (SCOPE=54).

-- Component-names appear to be qualified when referenced. This naming convention allows unidentified component-names in the case where the target dataset-name of a SUBSET is not invoked.

-- When duplicate component-names exist then the OCUR of the last of the duplicates will be returned (e.g., LASTNAME appears at OCUR=8 and OCUR=25).

-- DDL will guarantee the uniqueness of component-names in the DATABASE.

-- The INVOKE.NUMBER starts at one and is bumped for each INVOKE specified.

-- The PxTH.NO is assigned by sequentially searching the PATH.DICTIONARY.TABLE for a matching structure number. If a match is not found then the new structure number is added to the end of the table. The index into the table is the PATH.NO. Note that the DDL-NUMBER in the example contains only the STRUCTURE.NO (the time and date part has been omitted for clarity).

PATH TABLE AND ALGORITHM

```
DECLARE

01  PATH.TABLE   REMAPS BASE,
      02 PATH.OCUR              BIT(12),
      02 LAST.ENTRY.LINK        BIT(24),
      02 PATH.TYPE              BIT(3),
      02 INVOKE.NUM             BIT(8),
      02 PATH.NUM               BIT(8),
      02 COMP.OCUR(MAX.COMP)    BIT(12),
01  TARGET.STACK  (16),
      02 TS.LEVEL               BIT(8),
```

02 TS.PATH BIT(8);

Find a PATHNAME — TXDN followed by "DATASET" or TXDN with level number 51-53. Stop on "WORKING-STORAGE SECTION" or "PROCEDURE DIVISION".

Fill in PATH.OCUR — Determine type from level number.

Fill in PATH.TYPE = Determine

LEVEL	TYPE
01	0 dataset-name
02-49	1 dataset-name (Nested)
51	2 order-name
52	3 retrieval-name
53	4 subset-name

If level=01, then current INVOKE.NUMBER gets bumped.

Fill in INVOKE.NUM — Fill in PATH.NUM = Determine from looking up DDL# in path dictionary; if not found, add new entry.

Adjust target stack — Pop stack until level on top is less than current level (or stack empty). If current level <50 (DATASET), then push new entry.

Fill in TRGT.PATH.NUM — If target DDL# is explicit, find from path dictionary, adding a new entry if necessary. If target is implicit, then use path on top of target stack.

Fill in an OCUR — For each component-name which follows in the parenthesized list. Add an extra OCUR of zero, terminating the list.

INTERMEDIATE FILE TOKENS

ALLFILE AND ONFILE TOKENS

00=TINTGR	Integer: e.g., 123
00=TEDC	ENV.DIV.BLOCK code (ONFILE only)
01=TREAL	Real number: e.g., 12.3
01=TDDC	DATA.DIVISION.BLOCK code (ONFILE only)
02=TNNLIT	Non-numeric literal:e.g., "ABC"
02=TPDC	PROC.DIV.BLOCK code (ONFILE only)
03=TXDN	Explicit data-name: e.g, 01 TXDN.
03=TFILEREC	E.g., READ FILEREC; created in EXPLODE.
04=TXSECN	Explicit section-name: e.g., TXSECN SECTION.
05=TXPAR	Explicit paragraph-name: e.g., TXPAR.
06=TFIXUP	FIXUP the indicated OCUR according to the code.
07=TWORD	Word e.g., MOVE 1 TO TWORD.
08=TWMON	MONITOR word operand
09=TCSPL	Corresponding left:e.g, MOVE CORR TCSPL TO TCSPR
10=TCSPR	Corresponding right: e.g, MOVE CORR TCSPL TO TCSPR
11=TQUAL	Word qualifier: e.g, MOVE 1 TO WORD OF TQUAL
12=TLABL	Label: e.g., GO TO TLABL.
13=TALTER	ALTER(ed) operand: e.g, ALTER TALTER TO PROCEED TO P2.
14=TPERF	PERFORM operand: e.g., PERFORM TPERF.
15=TPTHRU	PERFORM THRU operand: e.g., PERFORM P1 THRU TPTHRU.
16=TLMON	MONITOR label operand
17=TLQUAL	Label qualifier: e.g., GO TO P1 OF TLQUAL
18=TRESWD	Reserved word EG: GO, MOVE, ADD.
19=TEOB	End-of-block marker: for unblocking.
20=TEOF	End-of-file marker: for unblocking
21=TDOT	Period
22=TPC	PICTURE
23=TDOLAR	DOLLAR card
24=TERROR	Error or warning message
25=TCARD	Card marker, i.e., source card encountered
26=TPROCESSED	Processed Dictionary entries (ONFILE only)
27=TRDFNS	REDEFINES/RENAMES operand: e.g, 02 X REDEFINES TRDFNS
28=TLABELREC	Operand of "LABEL RECORDS ARE" clause
29=TCSPS	Corresponding sentinel: delimits a TCSPL,TCSPR pair
30=TSTATESTOP	Statement delimiter
31=TINDEX	INDEXED BY operand: e.g, 02 DN INDEXED BY TINDEX
32=TSUBS	Subscript for previous operand..created in EXPLODE
33=TBOOSTOP	Boolean primary delimiter..created in PROSYN
34=TARITHOP	Arithmetic operator: e.g., ADDO, DIV3
35=TINDSECT	For initializing ALTERed GO TO paragraphs of independent sections
36=TMINIMUM	Establishes minimum scale and left part for intermediate results of an arithmetic expression

SEGFILE TOKENS

00=TCOPX	COP table index...if 0 then inline TCOP follows.
01=TLIT	Literal of 4-bit or 8-bit units.
02=TCOP	Inline COP
03=TDADDR	Address eventually relative to DSEG0
04=TOP	S-language operator
05=TBADDR	LABEL.TABLE OCUR changed into branch address by FIXUP
06=TBITS	Bit string to insert into code...general use.
07=TSUBORINX	Inline COP for subscripts or indexes
08=TSEGLINK	Links sections belonging to the same code segment
09=TFACTOR	Factor for subscripted inline COPs
10=TBOUND	Bound for subscripted inline COPs
11=TALTERINDEX	Pointer relative to ALTER table base
12=TOPNDX	Sets LIT.FLAG to 0...COP table index
13=TLITFIX	Make a literal relative to a certain base(for I/O use)
14=TSTOP	Signals change in code segments
15=	Unused
16=	Unused
17=TMEMVALUE	For FIXUP...initializes data segment value in CODEFILE
18=TCARDADR	Data address to printout with card (created in DATSYN)
19=TEOB	End-of-block marker: for unblocking
20=TEOF	End-of-file marker: for unblocking
23=TDOLLAR	DOLLAR card
24=TERROR	Error or warning message
25=TCARD	Card marker, i.e., source card encountered
26=TCOMMLITFIX	Make a literal relative to a certain base(for I/O use)

MISCELLANEOUS

Note:
URS = Field for any use.
** = Type not present in this pass.
-- = Type permanently dropped in previous pass.

USAGE

000=RESERVED
001=CMP
010=DISPLAY
011=MIXED
100=ASCII
101=INDEX
110=UNDEFINED
111=CMP-3

DATA NAME FLAGS (DNFLAGS) (10)

REFERENCE COUNT	LABEL RECORD OPERAND	SORT FILE NAME	GROUP FLAG	NOT A CORR OPND	COND NAME	FILLER	MONITORED	FILE NAME
(2)	(1)	(1)	(1)	(1)	(1)	(1)	(1)	(1)

REFERENCE COUNT(to minimize number of COP entries):
0:NO REFERENCES
1:ONE REFERENCE
2:TWO OR MORE REFERENCES

LABEL FLAGS (6)

THIS SECT CONTAINS ALTERED GO TO PAR	MONITORED	OBJECT OF AT LEAST ONE ALTER	PERFORMED TERMINAL PARAGRAPH	EXPLICIT GO TO PAR	49FLAG
(1)*	(1)	(1)	(1)	(1)	(1)*

* On in section and all its paragraphs.
49FLAG:1=PRIORITY GTR 49
Explicit GO TO PAR flag is set in QUALIFICATION RESOLUTION (III)

URS IN LITERAL TOKENS (4)

FORMAT	ALL	SIGNED	BINARY
(1)	(1)	(1)	(1)

FORMAT:0=8BIT, 1=4BIT BINARY:1=LIT IS 24BIT BINARY

BASIC COP (53)

ADDRESS (31)	LENGTH (LOGICAL SIZE) (18)	SUB FLAG (1) MULTI-ENTRY	DATA TYPE (2)	ASCII FLAG (1)

DATA TYPE
---- ----
00=UNSIGNED 4 BIT
01=UNSIGNED 8 BIT
10=SIGNED 4 BIT
11=SIGNED 8 BIT

COP FACTORS (80)

#SUBS (2)	FACTOR1 (18)	FACTOR2 (18)	FACTOR3 (18)	SUBSCRIPT BOUNDS (24)

FPB INFO (80)

UNUSED (26)	FPB INDEX (9)	FILE ORG. (2)	FILE LABEL TYPE (4)	FILE HARDWARE (7)	FILE ACCESS (4)	FILE LIMITS FLAG (1)	FPB NUMBER (9)	FILE MAX REC LEN (18)

PCINFO (74)

MASK ADR (24)	MAPSZ (CORE SZ) (18)	SCALE (F)	SIGN (3)	CLASS (3)	BZ (1)	CP (1)

SIGN

0=U
1=S
2=J
3=K
4=E

CLASS

0=NUMERIC INTEGER
1=NUMERIC REAL
2=NE
3=AN
4=AB
5=AE

MON INFO (30)

MONITOR SYMBOL ADR. (24)	MON SYM LGTH (6)

ALLFILE: FROM INITIAL PARSING (I)

```
00=TINTGR                (29+(LENGTH*8))
 ------------------------------------------------
  TYPE   COLUMN   SCALE   BYTE LENGTH   EBCDIC LITERAL
  (6)    (7)      (8)     (8)           (255 MAX)
 ------------------------------------------------
01=TREAL                 SEE 00=TINTGR
02=TNNLIT                SEE 00=TINTGR
03=TXDN                  (13)
 --------------
  TYPE   COLUMN
  (6)    (7)
 --------------
04=TXSECN                **  % SECTIONS ARE MARKED AS SUCH ON DNFILE BUT
                             % AS TXPAR ON ALLFILE
05=TXPAR                 SEE 03=TXDN
06=TFIXUP                **
07=TWORD                 SEE 03=TXDN
08=TWMON                 SEE 03=TXDN
09=TCSPL                 SEE 03=TXDN
10=TCSPR                 SEE 03=TXDN
11=TQUAL                 **
12=TLABL                 SEE 03=TXDN
13=TALTER                SEE 03=TXDN
14=TPERF                 SEE 03=TXDN
15=TPTHRU                SEE 03=TXDN
16=TLMON                 SEE 03=TXDN
17=TLQUAL                **
18=TRESWD                (29)
 ----------------------------
  TYPE   COLUMN   KEY (IN HEX)
  (6)    (7)      (16)
 ----------------------------

19=TEOB                  (6)
 ------
  TYPE
  (6)
 ------
20=TEOF                  SEE 19=TEOB
21=TDOT                  SEE 03=TXDN
22=TPC                   SEE 03=TXDN
23=TDOLAR                (36)
 --------------
  TYPE   FLAGS
  (6)    (30)
 --------------
24=TERROR                (24)
 ----------------------------
  TYPE   COLUMN   WARN   ERROR#
  (6)    (7)      (1)    (10)
 ----------------------------
```

25=TCARD	SEE 19=TEOB
26=TPROCESSED	**
27=TRDFNS	SEE 03=TXDN
28=TLABELREC	SEE 03=TXDN
29=TCSPS	**
30=TSTATESTOP	**
31=TINDEX	**
32=TSUBS	**
33=TBOOSTOP	**
34=TARITHOP	**
35=TINDSECT	**

DNFILE: FROM INITIAL PARSING (II)

```
00=TEDC                 (6)
 ------
  TYPE
  (6)
 ------
01=TDDC                 SEE 00=TEDC
02=TPDC                 SEE 00=TEDC
03=TXDN                 (58+(LENGTH*8))
 ------------------------------------------------------------------------
  TYPE  LEVEL  DNFLAGS  DNOCUR  STACKHEAD#  SYMBOL LENGTH  SYMBOLS
  (6)   (16)   (10)     (14)    (7)          (5)            (30 MAX)
 ------------------------------------------------------------------------
  NOTE : LEVEL=2 CHARACTERS

04=TXSECN               (54+(LENGTH*8))
 ------------------------------------------------------------------------
  TYPE  49    SEG#  CURRENT  LABEL  STACKHEAD#  SYMBOL LENGTH  SYMBOLS
        FLAG        SECTION  OCUR
                    OCUR
  (6)   (1)   (7)   (14)     (14)   (7)          (5)            (30 MAX)
 ------------------------------------------------------------------------
  SEG# IS SET BY THIS PASS..SYMBOL SCRAMBLE DONE BY THIS PASS

05=TXPAR                SEE 04=TXSECN
06=TFIXUP               (25)
 ------------------------
  TYPE  OCUR  FIXUP CODE
  (6)   (14)  (5)
 ------------------------
  MARK SUCH THINGS AS:
    A. NOT CORRESPONDING OPERAND.
    B. GO TO PAR.

07=TWORD                (18+(LENGTH*8))
 ------------------------------------------
  TYPE  STACKHEAD#  SYMBOL LENGTH  SYMBOLS
  (6)   (7)         (5)            (30 MAX)
 ------------------------------------------
08=TWMON                SEE 07=TWORD
09=TCSPL                SEE 07=TWORD
10=TCSPR                SEE 07=TWORD
11=TQUAL                SEE 07=TWORD
12=TLABL                SEE 07=TWORD
13=TALTER               SEE 07=TWORD
14=TPERF                SEE 07=TWORD
15=TPTHRU               SEE 07=TWORD
16=TLMON                SEE 07=TWORD
17=TLQUAL               SEE 07=TWORD
18=TRESWD               **
```

```
19=TEOB                 (6)
 ------
  TYPE
  (6)
 ------
20=TEOF                 SEE 19=TEOB
21=TDOT                 **
22=TPC                  (32+(LENGTH*8))
 ------------------------------------------------------
  TYPE   PC OCUR   STACKHEAD#   SYMBOL LENGTH   SYMBOLS
  (6)    (14)      (7)          (5)             (32 MAX)
 ------------------------------------------------------
```

A blank is included at the end of the PICTURE to provide a convenient delimiter for PICTURE PROCESSOR and to keep PICTUREs unique from labe names in dictionary processing pass.

```
23=TDOLAR               (36)
 -------------
  TYPE  FLAGS
  (6)   (30)
 -------------
24=TERROR               **
25=TCARD                **
26=TPROCESSED           **
27=TRDFNS               SEE 07=TWORD
28=TLABELREC            SEE 07=TWORD
29=TCSPS                SEE 00=TEOC
30=TSTATESTOP           **
31=TINDEX               **
32=TSUBS                **
33=TBOOSTOP             **
34=TARITHOP             **
35=TINDSECT             **
```

ONEILE: FROM DICTIONARY PROCESSING (II)

```
00=TEDC                 (6)
 ------
  TYPE
  (6)
 ------
01=TDDC                 SEE 00=TEDC
02=TPDC                 SEE 00=TEDC
03=TXDN                 (46)
 ---------------------------------------
  TYPE  LEVEL  DNFLAGS  SAMENAME OCUR
  (6)   (16)   (10)     (14)
 ---------------------------------------
  FILLER: FILLER FLAG=1
  WORKING-STORAGE SECTION: ENTRY=0

04=TXSECN               (42)
 -------------------------------------------------
  TYPE  49FLAG  SEG#  SECTION OCUR  SAMENAME OCUR
  (6)   (1)     (7)   (14)          (14)
 -------------------------------------------------
05=TXPAR                SEE 04=TXSECN
06=TFIXUP               (25)
 ------------------------
  TYPE  OCUR  FIXUP CODE
  (6)   (14)  (5)
 ------------------------
07=TWORD                (20)
 ----------------------
  TYPE  UNRESOLVED OCUR
  (6)   (14)
 ----------------------
08=TWMON                SEE 07=TWORD
09=TCSPL                SEE 07=TWORD
10=TCSPR                SEE 07=TWORD
11=TQUAL                SEE 07=TWORD
12=TLABL                SEE 07=TWORD
13=TALTER               SEE 07=TWORD
14=TPERF                SEE 07=TWORD
15=TPTHRU               SEE 07=TWORD
16=TLMON                SEE 07=TWORD
17=TLQUAL               SEE 07=TWORD
18=TRESWD               **
19=TEOB                 SEE 00=TEDC
20=TEOF                 SEE 00=TEDC
21=TDOT                 **
22=TPC                  (25+(LENGTH+8))
 ----------------------------
  TYPE  OCUR  LENGTH  SYMBOLS
  (6)   (14)  (5)     (31 MAX)
 ----------------------------
```

Pictures have been looked up and now OCUR = unique picture. Symbols are retained for convenience for picture processing pass.

```
23=TDOLAR              (36)
 -------------
  TYPE  FLAG$
  (6)   (30)
 -------------
24=TERROR              **
25=TCARD               **
26=TPROCESSED          (26)
 -------------
  TYPE  #BITS
  (6)   (20)
 -------------
```

At beginning of each buffer - used to indicate how much data was previously processed in this iterative pass (which allows the dictionary to overflow) - this type is dropped in the next pass.

```
27=TRDFNS              SEE 07=TWORD
28=TLABELREC           SEE 07=TWORD
29=TCSPS               SEE 00=TEDC
30=TSTATESTOP          **
31=TINDEX              **
32=TSUBS               **
33=TBOOSTOP            **
34=TARITHOP            **
35=TINOSECT            **
```

DNFILE: FROM QUALIFICATION RESOLUTION (III)

```
00=TEOC                 DROPPED
01=TODC                 (6)
 ------
  TYPE
  (6)
 ------
02=TPDC                 DROPPED
03=TXDN                 (60)
 ------------------------------------------
  TYPE  LEVEL  DNFLAGS  SAMENAME OCUR  SCOPE
  (6)   (16)   (10)     (14)           (14)
 ------------------------------------------
  DNFLAGS REF.CNT has been updated.

04=TXSECN               (33)
 ----------------------------------
  TYPE  LABEL FLAGS  SEG#  SAMENAME OCUR
  (6)   (6)          (7)   (14)
 ----------------------------------

05=TXPAR                SEE 04=TSECN
06=TFIXUP               DROPPED
07=TWORD                (30)
 ------------------------------
  TYPE  RESOLVED OCUR  DNFLAGS
  (6)   (14)           (10)
 ------------------------------
  DNFLAGS needed for actions like special processing of condition names

08=TWMON                SEE 07=TWORD
09=TCSPL                SEE 07=TWORD
10=TCSPR                SEE 07=TWORD
11=TQUAL                DROPPED
12=TLABL                (47)
 ------------------------------------------------------
  TYPE  LABEL FLAGS  SEG#  SECTION OCUR  RESOLVED OCUR
  (6)   (6)          (7)   (14)          (14)
 ------------------------------------------------------

13=TALTER               (55)
 ------------------------------------------------------------------
  TYPE  LABEL FLAGS  SEG#  SECTION OCUR  RESOLVED OCUR  OBJECT TIME
                                                       ALTER INDEX
  (6)   (6)          (7)   (14)          (14)          (8)*
 ------------------------------------------------------------------
```

```
14=TPERF                (55)
 ------------------------------------------------------------------------
  TYPE  LABEL FLAGS  SEG#  SECTION OCUR  RESOLVED OCUR  TERM PAR #
  (6)   (6)          (7)   (14)          (14)           (8)
 ------------------------------------------------------------------------
  TERM PAR# CANNOT BE THE OCUR BECAUSE OF 8 BIT LIMIT

15=TPTHRU               SEE 14=TPERF
16=TLMON                SEE 12=TLABL
17=TLQUAL               DROPPED
18=TRESWD               **
19=TEOB                 SEE 01=TDOC
20=TEOF                 SEE 01=TDOC
21=TDOT                 **
22=TPC                  (25+(LENGTH*8))
 ------------------------------
  TYPE  OCUR  LENGTH  SYMBOLS
  (6)   (14)  (5)     (31 MAX)
 ------------------------------
23=TDOLAR               DROPPED
24=TERROR               (17)
 --------------------
  TYPE  WARN  ERROR#
  (6)   (1)   (10)
 --------------------
25=TCARD                **
26=TPROCESSED           DROPPED
27=TRDFNS               (58)
 ------------------------------------------------------
  TYPE  RESOLVED OCUR  DNFLAGS  SAMENAME OCUR  SCOPE
  (6)   (14)           (10)     (14)           (14)
 ------------------------------------------------------

28=TLABELREC            SEE 07=TWORD
29=TCSPS                SEE 01=TDOC
30=TSTATESTOP           **
31=TINDEX               **
32=TSUBS                **
33=TBOOSTOP             **
34=TARITHOP             **
35=TINDSECT             **
```

PCINFO: FROM MERGE (IV)

(93)

MASK ADDR (24)	MAPSZ (CORE SZ) (18)	LENGTH (LOGICAL SZ) (18)	SCALE (F)	SIGN (3)	CLASS (3)	BZ (1)	CP (1)	ERROR FLAG (1)

MAPSZ=CORE SIZE..LOGICAL SIZE= # units to accept from source..
If ERRORFLAG=1 then MASK ADR. = error# and SCALE = card column..
MASK is in constant pool..LENGTH later becomes part of basic COP..
Other info later becomes PCINFO portion of elementary token.

SIGN

0=US UNSIGNED (4 OR 8 BIT)
1=SS HIGH ORDER DIGIT (4 OR 8 BIT)
2=JS HIGH ORDER DIGIT OF LAST UNIT (4 OR 8 BIT)
3=KS HIGH ORDER CHAR (8 BIT)
4=ES +,-,CR,DB (8 BIT)

CLASS

0=INTEGER	(4 OR 8 BIT)	PC 999
1=REAL	(4 OR 8 BIT)	PC 999V99 (NUMERIC SCALED)
2=NE	(8 BIT)	PC $$,$$$ OR 999.99+
3=AN	(8 BIT)	PC XX9A
4=AB	(8 BIT)	PC AAA or ABA
5=AE	(8 BIT)	PC XXBAA

ALLONEILE: FROM MERGE (IV)

00=TINTGR (33+(LENGTH*8))

TYPE (6)	COLUMN (7)	URS (4)	SCALE (8)	BYTE LENGTH (8)	EBCDIC LITERAL (255 MAX)

01=TREAL SEE 00=TINTGR
02=TNNLIT SEE 00=TINTGR
03=TXDN (67)

TYPE (6)	COLUMN (7)	LEVEL (16)	DNFLAGS (10)	SAMENAME OCUR (14)	SCOPE (14)

04-TXSECN (44)

TYPE (6)	COLUMN (7)	URS (4)	LABEL FLAGS (6)	SEG# (7)	SAMENAME OCUR (14)

If MONITOR FLAG=1 then follows a non-numeric literal token.

05=TXPAR SEE 04=TXSECN
06=TFIXUP --
07=TWORD (71)

TYPE (6)	COLUMN (7)	URS (4)	DNOCUR (14)	DNFLAGS (10)	MONITOR SYMBOL ADR. (24)	MON SYM LGTH (6)

08=TWMON DROPPED
09=TCSPL SEE 07=TWORD
10=TCSPR SEE 07=TWORD
11=TQUAL --
12=TLABL (58)

TYPE (6)	COLUMN (7)	URS (4)	LABEL FLAGS (6)	SEG# (7)	SECT OCUR (14)	LABEL OCUR (14)

URS:

DUMMY TLABL CREATED FOR A. GO TO. (1)	UNUSED (3)

13=TALTER (66)

TYPE (6)	COLUMN (7)	URS (4)	LABEL FLAGS (6)	SEG# (7)	SECT OCUR (14)	LABEL OCUR (14)	ALTER IN (8)

14=TPERF (66)

TYPE (6)	COLUMN (7)	URS (4)	LABEL FLAGS (6)	SEG# (7)	SECT OCUR (14)	LABEL OCUR (14)	TERM PAR (8)

```
15=TPTHRU               SEE 14=TPERF
16=TLMON                DROPPED
17=TLQUAL               --
18=TRESWD               (29)
 ----------------------------
  TYPE  COLUMN  KEY (IN HEX)
  (6)   (7)     (16)
 ----------------------------
19=TEOB                 (6)
 ------
  TYPE
  (6)
 ------
20=TEOF                 SEE 19=TEOB
21=TDOT                 (13)
 --------------
  TYPE  COLUMN
  (6)   (7)
 --------------
22=TPC                  (27)
 ------------------------
  TYPE  COLUMN  PC OCUR
  (6)   (7)     (14)
 ------------------------
23=TDOLAR               (36)
 -------------
  TYPE  FLAG$
  (6)   (30)
 -------------

24=TERROR               (24)
 ----------------------------
  TYPE  COLUMN  WARN  ERROR#
  (6)   (7)     (1)   (10)
 ----------------------------
25=TCARD                SEE 19=TEOB
26=TPROCESSED           --
27=TRDFNS               (69)
 ------------------------------------------------------------------
  TYPE  COLUMN  URS  RESOLVED OCUR  DNFLAGS  SNAM OCUR  SCOPE
  (6)   (7)     (4)  (14)           (10)     (14)       (14)
 ------------------------------------------------------------------
28=TLABELREC            SEE 07=TWORD
29 TCSPS                DROPPED
30=TSTATESTOP           **
31=TINDEX               **
32=TSUBS                **
33=TBOOSTOP             **
34=TARITHOP             **
35=TINDSECT             This token put on SEGFILE at specific location
                        awaiting PROSYN processing...See SEGFILE LAYOUT.
```

DNINFO: (INTERNAL FILE) DATA DIVISION (V)

(96)

LEVEL	JUST	BWZ	OCCURS VALUE	COPX	PC FLG	LGTH	ADR.	USAGE	#SUBS
(7)	(1)	(1)	(18)	(12)	(1)	(18)	(33)	(3)	(2)

IF PC.FLG THEN LGTH=PCINFO INDEX.
IF LEVEL=50 THEN LGTH=NON OCUR.
IF #SUBS NEQ 0 THEN SUBSCRIPTED & #SUBS=1,2,OR 3.

USAGE

0=UNDEFINED
1=CMP
2=DISPLAY
3=MIXED
4=ASCII
5=INDEX
6=UNDEFINED
7=CMP-3

ASCII FLAG
----- ----
0=EBCDIC
1=ASCII

IF ENTRY=0 THEN ENTRY=WORKING STORAGE ELSE ENTRY=FD:
(96)

LEV (7)	SD (1)	FD (1)	UN-USED (19)	FILE ORG. (2)	FPB INDEX (9)	01 OCUR (14)	FILE LABEL TYPE (4)	FILE HDWR (7)	FILE ACCESS (4)	FILE LIMITS FLAG (1)	FPB NUM (9)	MAX REC (18)

FILE ACCESS

0 = SEQUENTIAL
1 = RANDOM
2 = UNDEFINED
3 = DYNAMIC

FILE ORGANIZATION

0 = SEQUENTIAL
1 = INDEXED
2 = UNDEFINED
3 = UNDEFINED

ALLONEILE: FROM EXPLODE (VI)

00=TINTGR (33+(LENGTH*8))

TYPE (6)	COLUMN (7)	URS (4)	SCALE (8)	BYTE LENGTH (8)	EBCDIC LITERAL (255 MAX)

01=TREAL SEE 00=TINTGR
02=TNNLIT SEE 00=TINTGR
03=TFILEREC (281)

TYPE (6)	COL (7)	URS (4)	COPX (12)	BASIC COP (53)	FPB INFO (80)	PC INFO (74)	DN FLAGS (10)	JUST (1)	BWZ (1)	NON INFO (30)	USAGE (3)

04=TXSECN (44)

TYPE (6)	COLUMN (7)	URS (4)	LABEL FLAGS (6)	SEG# (7)	SAMENAME OCUR (14)

05=TXPAR SEE 04=TXSECN
06=TFIXUP --
07=TWORD (281)

TYPE (6)	COL (7)	URS (4)	COPX (12)	BASIC COP (53)	COP FACTS (80)	PC INFO (74)	DN FLAGS (10)	JUST (1)	BWZ (1)	NON INFO (30)	USAGE (3)

08=TWNON --
09=TCSPL SEE 07=TWORD
10=TCSPR SEE 07=TWORD
11=TQUAL --
12=TLABL (58)

TYPE (6)	COLUMN (7)	URS (4)	LABEL FLAGS (6)	SEG# (7)	SECTION OCUR (14)	LABEL OCUR (14)

13=TALTER (66)

TYPE (6)	COLUMN (7)	URS (4)	LABEL FLAGS (6)	SEG# (7)	SECTION OCUR (14)	LABEL OCUR (14)	ALTER TABLE INDEX (8)

14=TPERF (66)

TYPE (6)	COLUMN (7)	URS (4)	LABEL FLAGS (6)	SEG# (7)	SECTION OCUR (14)	LABEL OCUR (14)	TERM PAR# (8)

15=TPTHRU SEE 14=TPERF

```
16=TLNON                --
17=TLQUAL               --
18=TRESWD               (29)
 ---------------------------
  TYPE  COLUMN  KEY (IN HEX)
  (6)   (7)     (16)
 ---------------------------
19=TEOB
 ------
  TYPE
  (6)
 ------
20=TEOF                 SEE 19=TEOB
21=TDOT                 (13)
 --------------
  TYPE  COLUMN
  (6)   (7)
 --------------
22=TPC                  DROPPED
23=TDOLAR               (36)
 -------------
  TYPE  FLAGS
  (6)   (30)
 -------------

24=TERROR               (24)
 ---------------------------
  TYPE  COLUMN  WARN  ERROR#
  (6)   (7)     (1)   (10)
 ---------------------------

25=TCARD                SEE 19=TEOB
26=TPROCESSED           --
27=TRDFNS               --
28=TLABELREC            --
29=TCSPS                --
30=TSTATESTOP           **
31=TINDEX               SEE 07=TWORD (IMPLICIT LEVEL 50)
32=TSUB                 **
33=TBOOSTOP             **
34=TARITHOP             **
35=TINDSECT             **
```

ALLDNFILE: FROM PROCEDURE DIVISION SYNTAX CHECK (VII)

00=TINTGR (26+((LENGTH*SIGNED)*(4 OR 8)))

TYPE (6)	URS (4)	SCALE (8)	LENGTH (8)	LITERAL (255 MAX)

01=TREAL SEE 00=TINTGR
02=TNNLIT SEE 00=TINTGR
03=TFILEREC (271)

TYPE (6)	URS (4)	COPX (12)	BASIC COP (53)	FPB INFO (80)	PCINFO (74)	DNFLAGS (10)	JUST (1)	BWZ (1)	MON INFO (30)

04=TXSECN (23)

TYPE (6)	URS (4)	LABEL FLAGS (6)	SEG# (7)

Label OCUR need not be in token since it can be computed from a sequential explicit label counter. The alter table index can also be computed to give to the code generator.

05=TXPAR SEE 04=TXSECN
06=TFIXUP --
07=TWORD (271)

TYPE (6)	URS (4)	COPX (12)	BASIC COP (53)	COP FACTORS (80)	PCINFO (74)	DNFLAGS (10)	JUST (1)	BWZ (1)	MON INFO (30)

Note: For DATE, TIME, TODAYS-DATE, etc, a TWORD is created (internal to PROSYN) with SUBF=0, #SUBS NEQ 0, and FACTOR1 tells which: 1=DATE, 2=TIME, 3=TODAYS-DATE, 4=TODAYS-NAME

07=TWORD (Cont.)

URS

INDEXED.DN (1)	COP RELATIVE INDICATOR (2)	INDEX (1)

INDEXED.DN =This is an indexed DATANAME (DN INDEXED BY...)
COP relative indicator:
Set internal to CODE/GENERATOR
0=USE SEG# & DISP in basic COP
1=DISP is relative to reusable TRASH BASE DISP
INDEX = 77 C USAGE INDEX.

08=TWMON --
09=TCSPL --
10=TCSPR --

```
11=TQUAL                --
12=TLABL                (51)
 ------------------------------------------------------
  TYPE  URS  LABEL FLAGS  SEG#  SECTION OCUR  LABEL OCUR
  (6)   (4)  (6)          (7)   (14)          (14)
 ------------------------------------------------------
  Search routine in PROSYN uses low order bit of URS to tell condition
  routine in CODEGEN to use NEXT SENTENCE OCUR (NS.OCUR). NS.OCUR is set
  by search routine in CODEGEN.

13=TALTER               (59)
 ------------------------------------------------------------------
  TYPE  URS  LABEL FLAGS  SEG#  SECTION OCUR  LABEL OCUR  ALTER INDEX
  (6)   (4)  (6)          (7)   (14)          (14)        (8)
 ------------------------------------------------------------------
14=TPERF                (59)
 ------------------------------------------------------------------
  TYPE  URS  LABEL FLAGS  SEG#  SECTION OCUR  LABEL OCUR  TERM PAR#
  (6)   (4)  (6)          (7)   (14)          (14)        (8)
 ------------------------------------------------------------------
15=TPTHRU               DROPPED (ALL PERFORMS= 14=TPERF)
16=TLMON                --
17=TLQUAL               --

18=TRESWD               (30)
 ---------------
  TYPE  URS  KEY
  (6)   (8)  (16)
 ---------------
    If KEY=ALPHABETIC or NUMERIC then
    URS
   --------------------
    NOT CLASS  UNUSED
    (1)        (7)
   --------------------
    If KEY=RELATIONAL.OP (LSS,LEQ,NEQ,EQL,GEQ,GTR) then
    URS
   -------------------------
    UNUSED  IMPLIED SUBJECT
    (7)     (1)
   -------------------------
19=TEOB                 (6)
 ------
  TYPE
  (6)
 ------
20=TEOF                 SEE 19=TEOB
21=TDOT                 DROPPED (Changed to RESERVED)
22=TPC                  --
23=TDOLAR               (36)
 -------------
  TYPE  FLAGS
  (6)   (30)
 -------------
```

```
24=TERROR                (24)
 ---------------------------
  TYPE  COLUMN  WARN  ERROR#
  (6)   (7)     (1)   (10)
 ---------------------------
25=TCARD                 SEE 19=TEOB
26=TPROCESSED            --
27=TRDFNS                --
28=TLABELREC             --
29=TCSPS                 DROPPED

30=TSTATESTOP            SEE 19=TEOB
```

General use is to delimit a statement for CODEGEN because the CODEGEN control routine always does a get for a reserved word (unless INHIBIT.GET.OP is set). An example of use of TSTATESTOP: the MOVE verb may have multiple receiving fields and must keep scanning in PROSYN until it gets a non-receiving field token. Putting out the TSTATESTOP makes scanning in CODEGEN easier. also used in PROSYN by SORT, ACCEPT/DISPLAY, DUMP, SET, STOP RUN, and
Data Management.

```
31=TINDEX                SEE 07=TWORD   % IMPLICIT LEVEL 50
32=TSUB                  (75)
 ---------------------------
  TYPE  URS  COPX  BASIC COP
  (6)   (4)  (12)  (53)
 ---------------------------
33=TBOOSTOP              SEE 19=TEOB
34=TARITHOP              (19)
 ------------------------------------------------------------------
  TYPE  URS  OPERATOR  OPERATOR   ROUND  SIZE   MULTIPLE RECEIVING
                       CLASS             ERROR  FIELDS
  (6)   (4)  (4)       (2)        (1)    (1)    (1)
 ------------------------------------------------------------------
    IF DIV3 THEN:
    URS
   ---------------------------------------
    REMAINDER ROUNDED  REMAINDER  UNUSED
    (1)                (1)        (2)
   ---------------------------------------
    IF STORE THEN
    URS
   -----------------------------------------------
    UNUSED  EMIT.BOF(1) FOR ** OR / IN COMPUTE
    (3)     (1)
   -----------------------------------------------
    OPERATOR:
       0=ADD,1=SUB,2=:MUL,3=DIV,4=EXP,5=MOD,6=STORE,7=STOREMOD,
       8=DELETE-TOP-OF-STACK,9=CHANGE-SIGN
    OPERATOR CLASS:
       EG: ADD0,ADD1,ADD2,ADD3
```

SUB0,SUB1,SUB2,SUB3
DIV1,DIV2,DIV3,
MUL1,MUL2

35=TINDSECT (33)

TYPE (6)	TXPAR (8)	ALTER	INDEX	TLABL OCUR (12)	TLABL SEG (7)

36=TMINIMUM

TYPE (6)	URS (4)	MINIMUM SCALE (8)	MINIMUM LEFT PART (8)

Used by CODEGEN to establish minimum scale & left part for intermediate results

LABELIABLE: FROM CODEGEN (VIII).

(83 BIT ENTRY...17 PER DISK SEGMENT)

TYPE=3 (LINK)

TYPE (2)	UNUSED (64)	LABEL.OCUR (17)

TYPE=0,1,2 (0=PLAIN ADR,1=AND ADR,2=THEN ADR)

TYPE (2)	UNUSED (13)	SEG# (7)	CODE (22)	#BADDRS (13)	#LENBS (13)	#DISPS (13)	#DADDRS (13)

Note: #DISPS is count of DISPs that also have a SEG as part of the address. #DADDRs have DISP but no SEG.

SEGFILE: FROM ALL PRIOR PHASES

Note: MERGE, DATSYN, and CODEGEN each have a discrete area for their tokens. 18 = TCARDADR is emitted only by DATSYN. Most were designed for use by CODEGEN.

S E G F I L E H E A D E R (DISK SEGMENT 0...2 DISK SEGMENTS)

See compiler listing for contents.

P S E U D O C O D E D I C T I O N A R Y (11 DISK SEGMENTS)

(144 BIT ENTRY...10 PER DISK SEGMENT)

Used to collect code together as user specified.

LINK HEAD (20)	LINK TAIL (20)	UNUSED (30)	CODE (22)	#BADDRS (13)	#LENBS (13)	#DISPS (13)	#DADDRS (13)

T I N D S E C T E N T R Y T A B L E (1 DISK SEGMENT)

(10 BIT ENTRY...100 ENTRIES)

Number of entries for this code segment.

T I N D S E C T T O K E N S (100 DISK SEGMENTS)

(27 BIT ENTRY...53 PER DISK SEGMENT)

TXPAR (8)	ALTER INDEX	TLABL OCUR (12)	TLABL SEG# (7)

Note that this setup limits each user's independent code segment to 53 alterable GOTO paragraphs

---

P A T H D I C T I O N A R Y (DATA MANAGEMENT--12 disk segments)

M E R G E T O K E N S

See documentation on TOKENS

D A T S Y N T O K E N S

See documentation on TOKENS

D M D I C T T A B L E

(47 BIT ENTRY...30 PER DISK SEGMENT)

SEG (7)	DISP (21)	LENGTH (18)	ALLOCATE.DISK.ON.CODEFILE (1)

SEG=Data Dictionary Index
LENGTH=Byte length

Initialize: 1=WORKING STORAGE, FILE LABELS, etc.
0=Other (FD 01 RECORDS, etc.)
Last ONDICTTABLE entry is terminator set to all 0 bits.

C O P T A B L E
(55 BIT ENTRY...26 PER DISK SEGMENT)
BASIC COP (TYPE=0)

TYPE (2)	SEG (7)	DISP (24)	LENGTH (18)	SUBFI (1)	DTYPE (2)	ASCII (1)

NOTE:In final COP TABLE:
IF 1 SUB THEN 1 EXTRA COP TABLE ENTRY...
IF SUBF=1 THEN FOLLOWED BY
SUBSCRIPT ENTRY (TYPE=1)
IF 2 OR 3 SUBS THEN 2 EXTRA COP TABLE ENTRIES.

TYPE (2)	#SUBS (2)	UNUSED (1)	BOUND (24)	FACT1 (24)	UNUSED (2)

AND IF #SUBS>0 THEN FOLLOWED BY
SUBSCRIPT ENTRY #2 (TYPE=2)

TYPE (2)	UNUSED (3)	FACT2 (24)	FACT3 (24)	UNUSED (2)

F P B (2 DISK SEGMENTS)
One entry per file declared in the program.

FPB (AS MCP EXPECTS IT) (1 SEG)	OTHER FILE INFO: "FPB.TRAILER" (1 SEG)

A L T E R T A B L E
(21 BIT ENTRY...68 PER DISK SEGMENT)

TXPAR (7)	SEG	TLABL (14)	OCUR

Note: Arbitrary limit of 150 imposed by PARSE.

C O D E G E N T O K E N S
See below.

00=TCOPX (29)

TYPE (6)	COPX (12)	SUBFLAG (1)	#SUBS (2)	SUBBED.ON (1)	SEG# (7)

```
01=TLIT                 (16+((LENGTH+SIGN)*(4 OR 8)))
 ----------------------------
  TYPE  LTYPE  LGTH  LSYMB
  (6)   (2)    (8)   (255 MAX)
 ----------------------------
  LTYPE
  -----
  0=4 BIT UNSIGNED
  1=8 BIT
  2=4 BIT SIGNED
  3=RESERVED

02=TCOP (INLINE)        (64)
 ------------------------------------------------------------
  TYPE  BASIC COP  #SUBS  SUBBED.DN  COP RELATIVE INDICATOR
  (6)   (53)       (2)    (1)        (2)
 ------------------------------------------------------------
    SUBBED.DN:
        IF SUBF AND SUBBED.DN THEN SUBSCRIPTED DATANAME
        IF SUBF AND NOT SUBBED.DN THEN INDEXED DATANAME

03=TDADDR               (32)
 ----------------------------------------
  TYPE  TDADDR RELATIVE INDICATOR  DADDR
  (6)   (2)                        (24)
 ----------------------------------------
    TDADDR RELATIVE INDICATOR:
        00=RESERVED
        01=DATA SEG 0 RELATIVE
        10=RELATIVE TO TRASH

04=TOP                  (18)
 ---------------
  TYPE  OP CODE
  (6)   (12)
 ---------------
05=TBADDR               (39)
 ------------------------
  TYPE  LABEL.TABLE OCUR
  (6)   (33)
 ------------------------
06=TBITS                (14+LENGTH)
 -----------------------------
  TYPE  BIT LENGTH  BIT STRING
  (6)   (8)         (255 MAX)
 -----------------------------
07=TSUBORINX            (61)
 -----------------------------------
  TYPE  BASIC COP  COP.REL.INDICATOR
  (6)   (53)       (2)
 -----------------------------------
```

08=TSEGLINK (34)

TYPE (6)	DISK ADR OF NEXT SECTION TO BE PLACED IN THIS PHYSICAL CODE SEG (20)	SEG# (8)

09=TFACTOR (24)

TYPE (6)	FACTOR (18)

10=TBOUND (30)

TYPE (6)	TABLE BOUND (24)

11=TALTERINDEX (14)

TYPE (6)	ALTER INDEX (8)

Note: Size of container for ALTER table is not known. This is a pointer relative to ALTER table base(used by GOPAR operator).

12=TOPNDX SEE 00=TCOPX

13=TLITFIX (28+(LENGTH*(4 OR 8)))

TYPE (6)	LTYPE (2)	LENGTH (8)	BIT.DISP (10)	TDADDR RELATIVE INDICATOR (2)	LITERAL (160 MAX)

Note: LENGTH does not include BIT.DISP or TDADDR relative indicator. BIT.DISP is bit location in the LIT to be fixed up. The length of the area to be fixed up is assumed to be 24.
Note: For I/O type verbs the ADR is changed in FIXUP to be BASE REG. relative. i.e: include COP table & EDIT MASK.

14=TSTOP SEE 19=TE08

Note: Terminates a portion of code when CODEGEN encounters a TXSECN with SEG# NE0 SEG# of last TXSECN.

15 **

16 **

17=TMEMVALUE (70+(LENGTH*(4 or 8)))

TYPE (6)	ASCII FLAG (1)	LTYPE (1)	LIT LENGTH (8)	ALL FLAG (1)	RIGHT JUST (1)	DNDICT TABLE INDEX (10)	DIGIT DISP (24)	DEST SIZE (18)	LSYMB (160 MA

Note: Maps "VALUE" clause onto codefile so initial values are loaded at run time.
DNDICTTABLE INDEX of 0 indicates DSEG0

18=TCARDADR (51)

TYPE (6)	DNDICTTABLE INDEX (9)	DIGIT DISP (24)	COPX (12)

Note: To give an address associated with a card to printed on the report.

19=TEOB (6)

TYPE (6)

20=TEOF SEE 19=TEOB

23=TDOLAR (36)

TYPE (6)	FLAG$ (30)

24=TERROR (24)

TYPE (6)	COLUMN (7)	WARN (1)	ERROR# (10)

25=TCARD SEE 19=TEOB

26=TCOMMLITFIX (28+(LENGTH*(4 OR 8)))

TYPE (6)	LTYPE (2)	LENGTH (8)	FIX.CTNUM (10)	TDADDR (2)	RELATIVE INDICATOR	LITERAL (160 MAX)

Fields are analogous to those for TLITFIX except for FIX.CTNUM. The literal is assumed to be a communicate message in which CT.1 thru CT.10 may need to be fixed up. Each bit of FIX.CTNUM corresponds to a CT. When the I-th bit of FIX.CTNUM is on, FIXUP will fix the CT.I (the I-th CT).

CODEFILE: FROM FIXUP (IX).

DISK ADR	CONTENTS
0	PROGRAM PARAMETER BLOCK (PPB)
1	RUN STRUCTURE
2	DATA DICTIONARY
	DATA (THE VARIABLE DSEG0.DISK.ADR POINTS TO HERE)
	CODE DICTIONARY
	CODE
	FILE PARAMETER BLOCK(S) (FPB)
	PATH DICTIONARY (DATA MANAGEMENT)

INDEX