THE 501

COBOL

NARRATOR

NOVEMBER, 1960

Revised December, 1960

ACKNOWLEDGMENT

"This publication is based on the COBOL System developed in 1959 by a voluntary committee composed of government users and computer manufacturers. The organizations participating in the original development were:

Air Materiel Command, U. S. Air Force
Bureau of Standards, Department of Commerce
Datamatic Division, Minneapolis-Honeywell Corporation
David Taylor Model Basin, Bureau of Ships, U. S. Navy
ElectroData Division, Burroughs Corporation
International Business Machines Corporation
Radio Corporation of America
Remington-Rand Division of Sperry-Rand, Inc.
Sylvania Electric Products, Inc.

"The initial specifications for the COBOL language, printed by the Government Printing Office in 1960, are the result of contributions made by all of the above-mentioned organizations and no warranty expressed or implied, as to the accuracy and functioning of the programming system and language is made by any contributor or by the committee and no responsibility is assumed by any contributor or by the committee in connection therewith.

"It is reasonable to expect that many improvements and additions will be made to COBOL. Every effort will be made to insure that improvements and corrections will be made in an orderly fashion making proper provision not to invalidate existing users' investments in programming.

"The authors and copyright holders of the copyrighted material used herein:
FLOW-MATIC (Trade-mark of Sperry Rand Corporation) Programming for the UNIVAC ® I and II, Data Automation Systems © 1958, 1959, Sperry Rand Corporation; IBM Commercial Translator, Form No. F 28-8013, copyrighted 1959 by IBM, have specifically authorized the use of this material, in whole or in part, in the COBOL specifications. Such authorization extends to the reproduction and use of COBOL specifications in programming manuals or similar publications.

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

INDEX

COBOL Compatibility

The following comments discuss the compatibility available among the RCA 501, 301, and 601 COBOL Narrators. The reader is advised, however, that these general comments also embrace the area of compatibility involved with competitive COBOL compilers.

In general, "compatibility" refers to a program sharing ability by different computer systems. If, however, the code used in writing the program contains certain features not meaningful to the different computer systems, a transcription must be made of the original problem code into the order codes of the other computers. This is due solely because of the hardware-oriented, restrictive nature of machine code.

In particular, "compatibility", means the ability to take a COBOL program originally written for one system and recompile it to run on a different system. This is possible because of the substitution of a problem-oriented program description for the machine code description. The various Narrators serve as the transcription mechanism to create the order code which is recognizeable to each computer. Because COBOL is not totally problem-oriented, certain modifications are necessary to accommodate the machine dependent features of a COBOL program description.

To further classify the problems of COBOL compatibility, consider the following two categories:

1. Language content
2. Language structure

The first category concerns the problem-oriented elements of the language. Of the three major divisions within COBOL, Environment, Data, Procedure, only the Procedure Division may contain machine independent terms exclusively. This means that all the verbs in this division will operate in the identical manner on any machine having a COBOL processor. The one exception to this rule is the ENTER verb which contains a machine-oriented pseudo-code. Therefore, Procedure Divisions are compatible only after the ENTER statements are revised to reflect the proper pseudo-code for each computer upon which the pro- is to run. The Environment Division contains totally machine oriented terms and must therefore be rewritten in order to preserve compatibility. Information concerning the computer and any special names would not be applicable when transferring this program to another machine. For example, the statement, "OBJECT COMPUTER. 501." would have no validity when processed by the 601 COBOL Narrator. The files to be manipulated and the rerun information will still be valid such that the rewrite of this division to maintain compatibility is relatively small. The Data Division will require the most lengthy revision when it is desirable to change the memory mapping requirements to take advantage of the features of other computers. Organization of data is strictly machine oriented and must be revised dependent upon computer requirements. File names, record names, and item names will still be valid. However, the statement, "RECORDING MODE IS BLOCK" would have no significance to the 301 COBOL Narrator.

The second category concerns the structure of the language in relation to each individual implementation. This involves the problems of format and presence or absence of features. In order for a program written for one machine to be recompiled on another, the COBOL compiler used for recompilation must contain at least those elements included in that compiler which was used for the initial compilation.

Because it is anticipated that both the 301 and the 601 COBOL Narrators will contain all of those elements defined as Required COBOL, compatibility between these machines is affected primarily by computer capacities and problem-oriented language content. Thus, problems written in either 601 or 301 Required COBOL can be interchanged. Similarly, 501 programs can be compiled on either 301 or 601 because all 501 COBOL features are either identical to those in 301 and 601 COBOL or are a logical subset of those features.

To compile 301 and 601 COBOL programs on the 501 requires a similar investigation to determine if all features referenced with a 301 or 601 COBOL program are included in the 501 compiler.

Currently, the COBOL Maintenance Committee is in a special session devoted to the definition of those features to be labeled as "Required COBOL-1961". The completion of this task will prescribe the final structure of both 301 and 601 COBOL as well as determining the extent of revision necessary to fashion a compatible 501 compiler.

501 NARRATOR

INTRODUCTION

The 501 Narrator is an automatic coding system designed for the RCA 501 computer. The salient feature of the Narrator is that it is a problem-oriented system, whereby data processing applications may be expressed in natural English statements, rather than in the restricted language of a computer. The primary source of appeal is intended for those not familiar with machine coding; consequently, the language reflects terms common to business applications, rather than to computing systems.

Basically, the Narrator is an English Language compiler that accepts a series of English statements and translates them into a 501 machine-coded program. While it is admitted that complete "naturalness" cannot always be achieved, the user will find that the language offered with this compiler greatly simplifies procedures and reduces the time required to describe and code general business-type problems.

The development of the RCA Narrator was conducted in coincidence with a Department of Defense committee organized in May, 1959, to specify a common business language (COBOL). This committee was comprised of major computer manufacturers and users, and its mission was "to consider the development of specifications for a common business language for automatic programming of data processing systems." It was felt that the major needs for a common language were:

a. Present programming methods are time-consuming and costly.

b. Present methods do not permit rapid responce to changing problem requirements.

c. Manual reprogramming to shift applications to another make or model computer causes unacceptable delays.

The 501 Narrator is designed to alleviate these deficiencies by incorporating those COBOL features that allow a proper and efficient description of problems and, at the same time, reflect the features of machine design.

NOTATION USED IN THIS MANUAL

Special notation has been used throughout this manual to describe the 501 Narrator format. The reader should become thoroughly familiar with their meanings:

1. All upper case words that are UNDERLINED are considered as KEY words. These words must be present and correctly spelled, otherwise, an error will occur.

   Example: $\underline{\text{MULTIPLE}}$ $\underline{\text{REEL}}$

2. All upper case words that are NOT underlined are considered as OPTIONAL words. These words have been included in the language to facilitate readability; they may be used or omitted at the programmer's discretion. However, when OPTIONAL words are used they must be correctly spelled, otherwise, an error will occur.

   Example: $\underline{\text{POINT}}$ LOCATION IS $\underline{\text{LEFT}}$

3. All lower case words represent generic terms whose value must be supplied by the user.

   Example: $\underline{\text{ALTER}}$ line-no $\underline{\text{TO}}$ $\underline{\text{PROCEED}}$ $\underline{\text{TO}}$ line-no

4. Information within braces indicates that a CHOICE must be made.

   Example: $\underline{\text{DISPLAY}} \left\{ \begin{array}{l} \text{Data-name} \\ \text{literal} \end{array} \right\} \underline{\text{UPON}}\ \underline{\text{MONITOR}}.$

5. Information within square brackets represents an OPTION that may be included or omitted at the user's choice.

   Example: $\underline{\text{PERFORM}}\ \text{line-no} \left[\underline{\text{THRU}}\ \text{line-no} \right] .$

PUNCTUATION

The following rules govern the use of punctuation characters:

. Period = must appear where indicated.
, Comma = non-critical character (may be used or omitted).
; Semicolon = non-critical character (may be used or omitted).
" Quotation Marks = quotation marks are used to surround literals; they may not be used otherwise.
(Left Parenthesis = may only be used to denote "subscripting" (see page 47), or as a character in a literal.
) Right Parenthesis = same as above.

Periods, commas, semicolons, ending quote and right parenthesis must be followed by a space.

WORDS

A Narrator "word" is composed of not more than 30 characters, and may be constructed from the following set:

Ø thru 9
A thru Z
- (hyphen)

Words are ended by spaces; or they are ended by a period, comma, semicolon or right parenthesis, followed by a space. Since the primary purpose of a "space" is to separate words, "spaces" or "blanks" cannot appear within words. It should be noted that the number of spaces between words is not critical.

Note: The only time that a "space" has significance is when it appears within a literal.

All references in this manual to condition-names, file-names, record-names, and data-names follow the above rules. In addition, all "names" must contain at least one alphabetic character (hyphen is not considered as alphabetic).

RESERVED WORDS

All words shown in this manual as KEY or OPTIONAL words may not be used for data-names. In addition, the following words have special significance, and may only be used as specified in this manual:

BEGINNING-TAPE-LABEL	REEL-NUMBER	ALL	ZEROS
ENDING-TAPE-LABEL	DATE-WRITTEN	FILLER	ZEROES
IDENTIFICATION	PURGE-DATE	TALLY	SPACE
ID	BLOCK-COUNT	ZERO	SPACES

HYPHENS

Hyphens may only appear within words and, as such, are always considered as being part of the word. It is not permissible to begin or end words with hyphens.

Example: MASTER-FILE, QUANTITY-ON-HAND

LITERALS

Literals used in the Narrator must be enclosed by quotation marks and may not exceed 120 characters. Literals may be composed from the following characters:

A through Z	, comma	(left parenthesis
Ø through 9	. period	) right parenthesis
$ dollar sign	; semicolon	# number sign
% percent sign	: colon	/ slant (see note)
+ space, or plus	- hyphen, or minus	* asterisk (see note)
		& ampersand (see note)

Examples: "END OF JOB", "1875", "DECEMBER 31, 196Ø."

Note: The slant, asterisk and ampersand symbols have special significance and may only be used in the CONSTANT SECTION--see page 58.

GENERAL DESCRIPTION

Input to the 501 NARRATOR consists of four divisions, which appear in the following order:

IDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
DATA DIVISION.
PROCEDURE DIVISION.

Before treating these divisions in detail, a capsule description of their functions follows.

Identification Division

The Identification Division contains descriptive information that identifies the program being compiled. This information will appear in the reference listing provided for each program.

Environment Division

In a sense, the purpose of the Environment Division is to describe to the compiler the "outside world." For example, the compiler must know the computer upon which the object program will be run and the equipment configuration that will be used -- tape stations, printer, etc. In addition, this Division also supplies information concerning re-run procedures and file selection.

Data Division

The Data Division furnishes the compiler with a description of the "inside world." It provides a detailed "memory-picture" of the files, records, working storage and constants that the object program will use and manipulate.

Procedure Division

This division consists of the steps -- expressed in meaningful English statements -- that the computer is to follow during the data processing operation. These statements are written as simple, imperative commands that include the concept of verbs to denote actions, sentences to describe procedures, and IF clauses to provide alternative paths of action.

If desired, the user may also incorporate into this division coding written in RCA 501 Automatic Assembly format.

IDENTIFICATION DIVISION

The IDENTIFICATION DIVISION is the first division of the NARRATOR input; it is used to provide descriptive information on the reference listing to identify this particular program. It is written in the following format:

IDENTIFICATION DIVISION.

PROGRAM-ID.	
AUTHOR.	(Optional)
INSTALLATION.	(Optional)
DATE-WRITTEN.	(Optional)
DATE-COMPILED.	(Optional)
SECURITY.	(Optional)
REMARKS.	(Optional)

EF

The PROGRAM-ID entry must always appear; other entries, if present, must appear in the order listed. The format for each entry is as follows:

PROGRAM-ID. program-name.

```
Examples: PROGRAM-ID.  PAYROLL.
          PROGRAM-ID.  INITIAL INVENTORY UPDATING RUN.
```

Information supplied for this entry may not exceed 30 characters, including spaces.

AUTHOR. author's name.

```
Examples: AUTHOR.  JOHN DOE.
          AUTHOR.  ROBERT T SMITH, JR.
```

Information supplied for this entry may not exceed 100 characters, including spaces.

INSTALLATION. installation name.

```
Examples: INSTALLATION.  ANY MFG CO CHICAGO BRANCH.
          INSTALLATION.  RCA RECORD DIVISION.
```

Information supplied for this entry may not exceed 100 characters, including spaces.

DATE-WRITTEN. date.

```
Examples: DATE-WRITTEN.  SEPTEMBER 23, 196Ø.
          DATE-WRITTEN.  12 MAY, 1961.
```

DATE-COMPILED. date.

Examples: DATE-COMPILED. DECEMBER 15, 196Ø.
DATE-COMPILED. TODAY.

Information supplied for this entry may not exceed 30 characters, including spaces.

If the word TODAY appears for the DATE-COMPILED entry, the Narrator will request (at compilation time) that the compilation date be supplied in a message from the Paper Tape Reader. The date in this message will then be used for the DATE-COMPILED entry. This information is punched as follows: < date >.

Example: < FEBRUARY 6, 1961 >

SECURITY. ....

Examples: SECURITY. CLASSIFIED.
SECURITY. RESTRICTED FOR ACCOUNTING PERSONNEL ONLY.

Information supplied for this entry may not exceed 100 characters, including spaces.

REMARKS. ....

Example: REMARKS. INPUT SUPPLIED BY ACCOUNTS RECEIVABLE RUN; OUTPUT TO BE USED FOR BILLING RUN.

Information supplied for this entry may not exceed 200 characters, including spaces.

Notes:

1. Periods are not to appear within the information supplied by the writer except to end the entry.

2. The IDENTIFICATION DIVISION is terminated by an EF symbol.

EXAMPLE

```
IDENTIFICATION DIVISION.

     PROGRAM-ID.  MONTHLY REPORT RUN.
     AUTHOR.  THOMAS W BROWN.
     DATE-COMPILED.  TODAY.
     REMARKS.  MASTER-ORDER-FILE AND BACK-ORDER-FILE REQUIRED AS INPUTS FOR
               THIS RUN.

          EF
```

The Environment Division is written in the following order:

```
ENVIRONMENT DIVISION.

CONFIGURATION SECTION.
     OBJECT-COMPUTER.
     SPECIAL-NAMES.  (optional)

INPUT-OUTPUT SECTION.
     FILE-CONTROL.
     I-O-CONTROL.  (optional)
```

EF

A. CONFIGURATION SECTION

The Configuration Section provides the compiler with a description of 1) the object computer, i.e., the computer upon which the final program is to be run, and 2) any Special Names which are given to computer breakpoint switches.

Format:

```
CONFIGURATION SECTION.

 OBJECT-COMPUTER.  5Ø1  [, MEMORY ADDRESS address-1 {THROUGH} address-2] .
                                                    {THRU   }

[SPECIAL-NAMES.  BREAKPOINT-number ON STATUS IS condition-name-1,

                [, BREAKPOINT-number etc. ---]. ]
```

MEMORY ADDRESS:

If the user does not specify MEMORY ADDRESS, the program will be compiled for memory locations 004000 through 037777.

If the user does specify MEMORY ADDRESS, the beginning address must end in 0 and should not be lower than 004000; the ending address must end in 7. It should be noted that if any file is assigned to the On-Line printer, the ending address must end in X77 (where X is an odd number).

Examples:

a) OBJECT-COMPUTER. 5Ø1.

b) OBJECT-COMPUTER. 5Ø1, MEMORY ADDRESS ØØ4ØØØ THRU 137777.

(Cont'd)

ENVIRONMENT DIVISION

The second division of the Narrator input -- the ENVIRONMENT DIVISION -- consists of two separate sections that provide the compiler with the following information:

CONFIGURATION SECTION	1. Memory area available for the object program.
	2. Use of computer Breakpoint Switches.
INPUT-OUTPUT SECTION	1. Peripheral equipment (tape stations, printer, etc.) to be utilized by various input and output files.
	2. Program restart requirements.
	3. Common file areas and alternate file area requirements.

SPECIAL-NAMES

The "Special-Names" statement is optional. If used, "BREAKPOINT-number" must appear as BREAKPOINT-1 through BREAKPOINT-5. (Breakpoint Ø is reserved for the Input/Output Control.)

"Condition-name" can be any unique name up to 30 characters; there must be a "condition-name" for each breakpoint specified.

Examples:

a) SPECIAL-NAMES. BREAKPOINT-3 ON STATUS IS WEEKLY-REPORTS.

b) SPECIAL-NAMES. BREAKPOINT-1 ON IS WEEKLY, BREAKPOINT-3 ON IS MONTHLY, BREAKPOINT-4 ON IS YEARLY.

The programmer may test the status of a particular breakpoint by referring to its condition-name in an IF statement in the PROCEDURE DIVISION. Thus, if breakpoint-1 had been assigned the condition-name"MONTHLY-SUMMARY," the on status of this breakpoint may be tested by the following statement:

```
IF MONTHLY-SUMMARY GO TO 214.
```

If breakpoint-1 is not set ("on"), the next Narrator statement will be executed.

EXAMPLE:

```
CONFIGURATION SECTION.
  OBJECT-COMPUTER.  5Ø1, MEMORY ADDRESS Ø1ØØØØ THRU Ø76377.
  SPECIAL-NAMES.  BREAKPOINT-5 ON STATUS IS CHECK-PRINT.
```

B. INPUT-OUTPUT SECTION

The Input-Output Section is composed of two parts— the File-Control and I-O-Control.

I. FILE CONTROL

A File-Control Section must appear in every program; the function of this section is to specify what tape stations or peripheral equipment will be used by the various files during the data processing operation. This section also informs the compiler if multiple-reel files are being used, and if alternate read/write areas are not desired.

```
INPUT-OUTPUT SECTION.
                                                   ⎧ PRINTER                    ⎫
FILE-CONTROL.                                      ⎪ TAPES t1, t2, t3, t4       ⎪
SELECT [OPTIONAL] file-name, ASSIGN               ⎨ integer TAPES              ⎬
                                                   ⎪ TCP integer TAPES          ⎪
                                                   ⎩ TCP TAPES t1, t2, t3, t4   ⎭

[, FOR MULTIPLE REEL]    [, RESERVE NO ALTERNATE AREAS] .
```

SELECT

1. A SELECT message must appear for each file used in the program.

2. The word "OPTIONAL" appearing after "SELECT" means that this file may or may not be present during the running program. (Note that only input files can be designated as OPTIONAL.) If an input file, designated as OPTIONAL, is not present at program running time, the Input-Output Control will transfer to the end-of-file routine when the first read command is encountered for that file. A subsequent read command for that file would cause an error.

3. The entry t1, ... t4 should appear in the following form and must be two digits.

   tn = 06, where 06 is the trunk number.

4. TCP represents Transcribing Card Punch. This entry appears when the output file is to be written to magnetic tape and subsequently to punched cards via the Transcribing Card Punch unit.

5. OUTPUT files cannot be assigned to tapes 76 and 77.

FOR MULTIPLE REEL

When a file may exceed one reel of tape, "MULTIPLE REEL" must be specified for that file.

If one tape station has been assigned to a file that has multiple reels, the object program will stop at the exhaustion of each reel so that a new reel may be mounted. When more than one tape station has been assigned, the Input-Output Control will automatically provide tape "swapping" without interrupting the program.

When intermediate reels are exhausted during the running object program, they will automatically be rewound to BTC. (If the user "closes" intermediate reels through a CLOSE REEL statement, the reel will be rewound unless "WITH NO REWIND" has been specified.)

ALTERNATE AREAS

In order to provide maximum simultaneity, the Narrator will assign alternate READ-IN areas for all INPUT files that are assigned to magnetic tape. In addition, if the records for a magnetic tape OUTPUT file are in message format, and not grouped (batched), an alternate WRITE area will be reserved for that file.

If the user does not desire alternate areas, "RESERVE NO ALTERNATE AREAS" must be specified.

Alternate file areas will not be reserved for files assigned to the Paper Tape Reader (tape 77), or to the On-line Printer (PRINTER). In these instances the clause RESERVE NO ALTERNATE AREAS is not required.

EXAMPLES:

```
a)   FILE-CONTROL.
       SELECT MASTER, ASSIGN 1 TAPE, FOR MULTIPLE REEL.
       SELECT TRANSACTIONS, ASSIGN 1 TAPE, RESERVE NO ALTERNATE AREAS.
       SELECT NEW-MASTER, ASSIGN 1 TAPE, FOR MULTIPLE REEL.

b)   FILE-CONTROL.
       SELECT MASTER, ASSIGN 2 TAPES, MULTIPLE REEL.
       SELECT TRANSACTIONS, ASSIGN 1 TAPE.
       SELECT SUMMARY-REPORTS, ASSIGN PRINTER.
       SELECT NEW-MASTER, ASSIGN 2 TAPES, FOR MULTIPLE REEL.
       SELECT BILLINGS, ASSIGN TCP TAPE Ø7, RESERVE NO ALTERNATE
         AREAS.
```

NOTES

1. The singular form of TAPES may be used if desired.

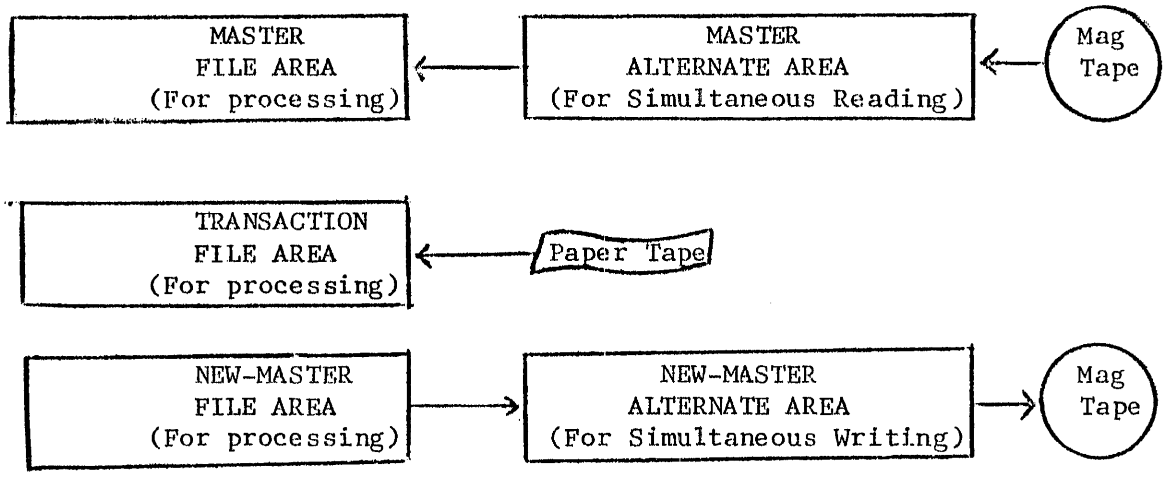
2. Tapes are automatically assigned to files in the order that the files appear under FILE-CONTROL. The order of assignment is 10, 20, 30, 40, 50, 60, 70, 00, 01, 02, 03, 04, 05, 06, 07, 11, 12, etc. If specific tape trunk assignments have been made by the programmer, these trunk numbers are eliminated from the above list, and are not reused for subsequent assignments.

Assignment of File and Alternate Areas

Assume that the Master-File (input), Transaction-File (input) and New-Master-File (output) contain single records in message format, and the programmer has defined the FILE CONTROL as follows:

```
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT MASTER-FILE, ASSIGN 1 TAPE.
    SELECT TRANSACTION-FILE, ASSIGN TAPE 77.
    SELECT NEW-MASTER-FILE, ASSIGN 1 TAPE.
```

The Narrator will reserve the following memory areas


MASTER FILE AREA (For processing)
MASTER ALTERNATE AREA (For Simultaneous Reading)
Mag Tape
TRANSACTION FILE AREA (For processing)
Paper Tape
NEW-MASTER FILE AREA (For processing)
NEW-MASTER ALTERNATE AREA (For Simultaneous Writing)
Mag Tape

During the running program, the Input/Output Control utilizes the Master Alternate area for bringing in records from the Master-File in the Simultaneous mode. When a READ command is executed for the Master-File, the record appearing in the Alternate area is transferred to the File area so that the programmer may process that record. (The next record from the Master-File is then brought into the Alternate area, awaiting transfer to the File area at the next READ command.)

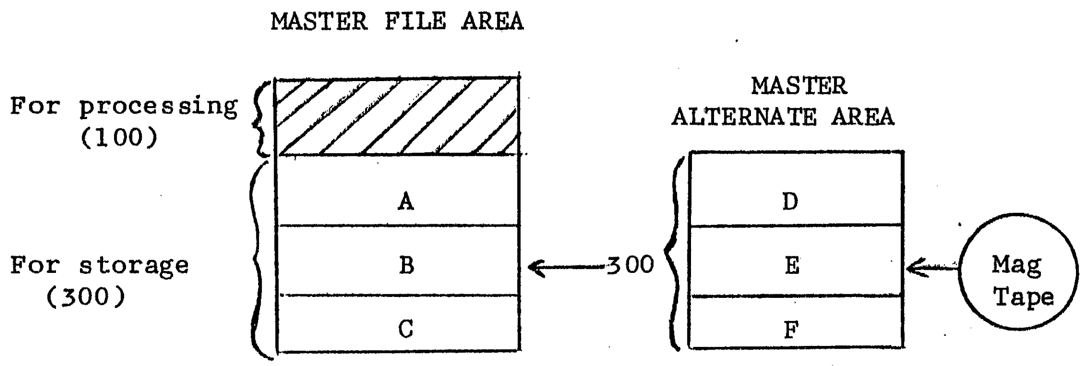
In the case of the Transaction-File (no alternate areas), the Input/Output Control reads directly into the File area each time a READ command is given for the Transaction-File.

When a WRITE command is given for the NEW-MASTER, the record appearing in the New-Master File area at that time is transferred by the Input/Output Control into the New-Master Alternate area. (This record will then be written to the output tape in the Simultaneous mode.) It should be remembered that the WRITE command does not place records into the File area -- this is the programmer's responsibility. Thus, if record "A" appears in the File area when a WRITE statement is performed, record "A" will be written out to the output file. If a subsequent WRITE command is given -- and another record has not been placed in the File area -- record "A" will again be written to the output file.

Generally, alternate areas are used with high-volume files. They should not be assigned for files that are relatively inactive, if the programmer wishes to make efficient use of computer memory.

"Batched" Input Files

If an Input File contains "batched" or "grouped" records, alternate areas are reserved somewhat differently. For example, assume that records in the Master-File appear in groups of three, and each record contains 100 characters:



The first group of records (A, B, C) is brought into the alternate area and transferred to the "storage" area. At the first READ command, record "A" (in the storage area) is placed into the processing area, and the programmer may now process that record. At the next READ command, record "B" is transferred into the "processing" area, and so on.

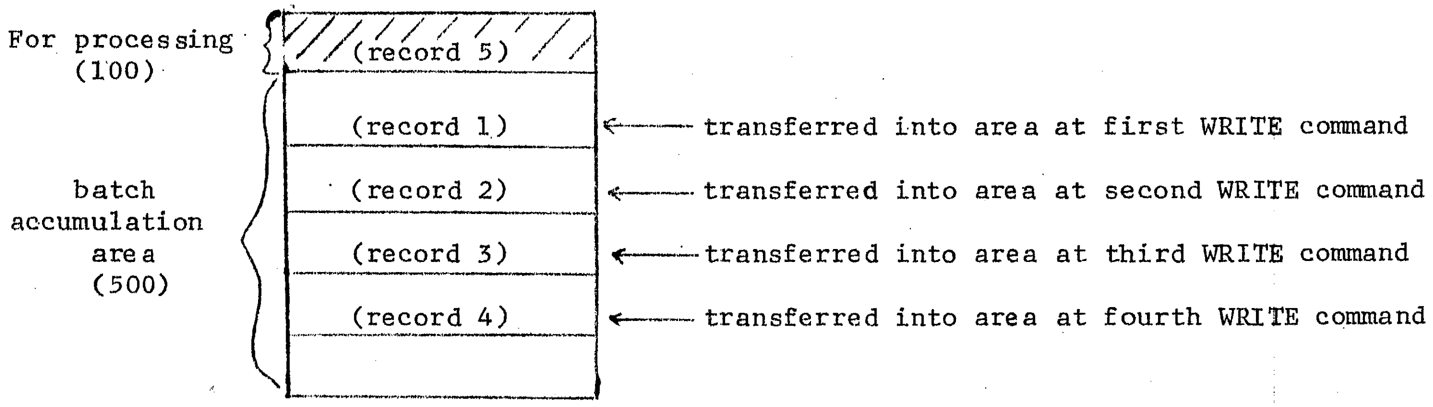
After the Input/Output Control has transferred record "C" (because of third READ command) into the processing area, the next group of records in the alternate area (D, E, F) are moved into the storage area. Thus, when the fourth READ command is given, record "D" will be placed into the processing area.

For programming purposes, then, the user processes "grouped" records as if they appeared as single records on the file. The actual reading in and internal transfer of records is automatically handled by the Input/Output Control.

"Batched" Output Files

If an OUTPUT file is "batched," the Input/Output Control accumulates records within the computer until a "batch" has been completed, at which time it will be written out to tape. If, for example, records in the output file are to appear in groups of five, the file area would appear as follows after the fourth WRITE command had been executed for that file:

OUTPUT FILE AREA


For processing (100)
(record 5)
batch accumulation area (500)
(record 1)
(record 2)
(record 3)
(record 4)
transferred into area at first WRITE command
transferred into area at second WRITE command
transferred into area at third WRITE command
transferred into area at fourth WRITE command

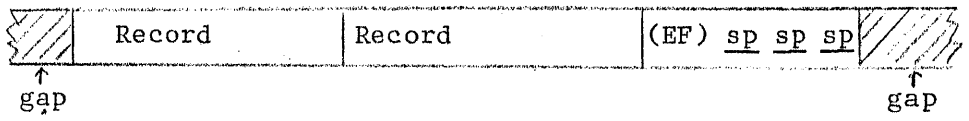
When the programmer issues the next WRITE command, record "5" will be transferred into the accumulation area and, since the "batch" is now complete, this group of records will be written out to the output file.

Here, again, the programmer releases records to the output file as if they appeared singly. The accumulation of these records into groups of five is automatically handled by the Input/Output Control.

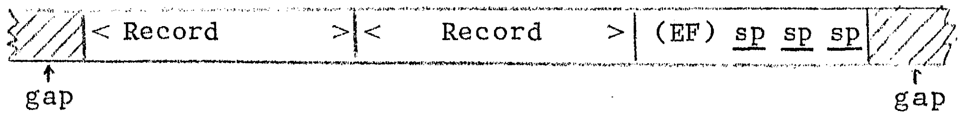
Important Note Regarding Last Batch of File

If the last batch in an INPUT file does not contain the maximum number of records, an EF symbol and three spaces must appear immediately following the last record. If, for example, records are grouped in batches of five, but there are only two records in the last batch, the last batch must appear as follows:

BLOCK FORMAT



RECORD FORMAT



For OUTPUT files, the Input/Output Control will automatically accommodate an incomplete last batch by inserting an EF symbol and three spaces immediately after the last record. Note that this only occurs when a batched output file is "closed."

II I-O-CONTROL

This part of the Input-Output Section is optional. When present it supplies the compiler with rerun instruction (if any), and specifies files which will use common memory areas (if any). It is written in the following format:

```
[ I-O-CONTROL.
[
      [       [ { ON file-name-1  } ]           { END OF REEL     }                  ]
      [ RERUN [ {                 } ]   EVERY   {                 } OF file-name-2 . ]
      [       [ { ON tape-number  } ]           { integer RECORDS }                  ]

      [ SAME AREA FOR file-name-3, file-name 4, .... . ]
```

RERUN OPTION

1. Only one RERUN statement may be given for a program.

2. Specifying RERUN in the Environment Division will bring about an automatic interruption in the running program at certain points. As each interruption occurs a memory dump will be taken and placed on the tape or file specified by the user; the program then continues until the next scheduled stop, and so on.

   Should it become necessary to restart the program, the user brings into memory the first block of the object program. (This block will contain the rerun program.) A paper tape call message is then prepared specifying the restart point desired and any new tape station assignments. Based on this message, the rerun program then automatically positions all input and output tapes, re-establishes memory, and continues the program from the specified restart point.

   Note: When a program is to be restarted the user will be advised, via the monitor printer, as to what files and tape stations are required for that rerun point. This permits restarting programs at any future time.

3. If a file-name or tape-number immediately follows the word "RERUN," the rerun information during the running program will be stored on this file or tape. If missing, the rerun information will be placed on the file-name specified at the end of the statement.

   Examples:

   a) RERUN ON 7Ø EVERY END OF REEL OF MASTER-FILE.

      Information for each restart point will be stored on tape 7Ø as follows:

      Tape 7Ø

Restart 1		Restart 2		Restart 3	etc.	

b) RERUN ON NEW-MASTER-FILE EVERY END OF REEL OF MASTER-FILE.

When the first restart point is reached (i.e. end of first reel of Master-File), the New-Master-File reel will be automatically "closed" and the restart information will be placed following the ED symbol. This procedure will occur for all subsequent reels.

c) RERUN EVERY END OF REEL OF NEW-MASTER-FILE.

Rerun information will be stored following the ED symbol on each reel of the New-Master-File.

Note: If the user selects a FILE to receive the rerun information, this file must

1. be a multiple-reel OUTPUT file.

2. have a beginning-tape-label so that the Narrator can identify it as the rerun file. In addition, both IDENTIFICATION (or ED) and REEL-NUMBER must appear in the label format.

3. since the rerun information will be placed on the output file following the ED symbol, sufficient magnetic tape must remain to contain this information. Approximately five feet of tape will be required for each module of memory involved.

4. If rerun is determined by "END OF REEL," the user may specify an OUTPUT file or a special work tape as the "receiver" of the rerun information.

Examples:

a) RERUN ON NEW-MASTER-FILE EVERY END OF REEL OF MASTER-FILE.

b) RERUN ON 7Ø EVERY END OF REEL OF MASTER-FILE.

If rerun is governed by a certain number of records, the "RERUN ON tape-number" option must be used. The tape number specified must be a "work" tape. This tape cannot be used elsewhere in the program, i.e., no File Description or SELECT.
Example:

RERUN ON 1Ø EVERY 2ØØØ RECORDS OF MASTER-FILE.

The maximum number of records for restart points is 99999.

5. "Tape-number" is written as Ø6, 3Ø, 7Ø, etc.

SAME AREA OPTION

In order to conserve computer memory, the Narrator user may indicate at compilation time that certain files are to share a common file area. This is accomplished through the SAME AREA option. For example:

a) SAME AREA FOR RECEIPTS, DISBURSEMENTS.

b) SAME AREA FOR NEW-MASTER, INVENTORY, SALES-SUMMARY. SAME AREA FOR RECEIPTS, DISBURSEMENTS.

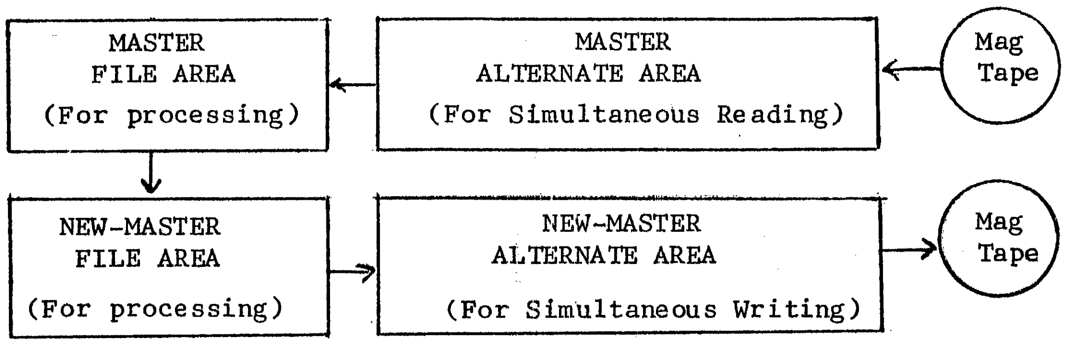
Notes:

1. A SAME statement must appear for each "set" or group of files that are to share the same memory area. A number of SAME statements may appear for a given program.

2. If two files are "batched" and are active at the same time (i.e., "opened"), they cannot share the same area.

3. Although all files designated in the SAME entry will share the same file area in memory, they will NOT share a common alternate area. This means, then, that any alternate areas established will not be affected by this entry.

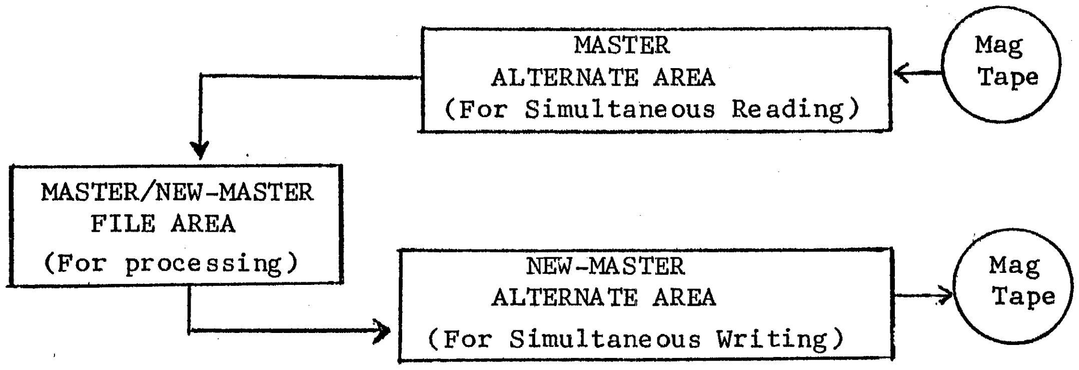
Example A: Master and New-Master file, both with alternate areas and not sharing the same area:



Note: On the "non-hit" path (i.e., when the Master file does not have to be processed) the programmer must first move the record from the Master file area to the New-Master file area before writing. For example:

```
MOVE MASTER TO NEW-MASTER.
WRITE NEW-MASTER.

        (or)

WRITE NEW-MASTER FROM MASTER.
```

Example B: Master and New-Master file, both with alternate areas, but sharing the same file area:



Note: On the "non-hit" path, the programmer may write the New-Master record directly from the file area, vis:

```
WRITE NEW-MASTER.
```

S-A-M-P-L-E

```
ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

  OBJECT-COMPUTER. 5Ø1.

  SPECIAL-NAMES. BREAKPOINT-5 ON STATUS IS WEEKLY-SUMMARY.

INPUT-OUTPUT SECTION.

  FILE-CONTROL.

    SELECT MASTER, ASSIGN 2 TAPES, MULTIPLE REEL.

    SELECT SALES, ASSIGN 1 TAPE.

    SELECT REPORT, ASSIGN PRINTER.

                  EF
```

S-A-M-P-L-E

```
ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

  OBJECT-COMPUTER. 5Ø1, MEMORY ADDRESS ØØ5ØØØ THRU 137777.

  SPECIAL-NAMES. BREAKPOINT-1 ON STATUS IS WEEKLY, BREAKPOINT-2 ON
  STATUS IS MONTHLY.

INPUT-OUTPUT SECTION.

  FILE-CONTROL.

    SELECT MASTER, ASSIGN 1 TAPE.

    SELECT OPTIONAL CHANGES, ASSIGN TAPE 7Ø, RESERVE NO ALTERNATE
    AREAS.

    SELECT SALES-REPORTS, ASSIGN 1 TAPE.

    SELECT ADDITIONS, ASSIGN 1 TAPE, RESERVE NO ALTERNATE AREAS.

    SELECT EXCEPTIONS, ASSIGN PRINTER.

  I-O-CONTROL.

    RERUN ON 1Ø EVERY 15ØØ RECORDS OF MASTER.

    SAME AREA ADDITIONS, CHANGES.

                  EF
```

DATA DIVISION

In the third division of the Narrator input -- the DATA DIVISION -- the user describes in detail the files and records that the object program will manipulate. In addition, the user also defines in this division working storage areas and constants required by the program. This information appears in the following order:

```
DATA DIVISION.
  FILE SECTION.
  WORKING-STORAGE SECTION. (Optional)
  CONSTANT SECTION. (optional)

                EF
```

The programmer should bear in mind that the generated machine-coded program must process records, data fields, etc., in strict adherence to the description of these areas as specified in the DATA DIVISION. Data descriptions and the organization of records,therefore, should be undertaken in a thorough manner if maximum efficiency in the running program is desired. For example, unnecessary editing requirements -- while not affecting the correctness of the object program - will, nevertheless, generate additional instructions in the object program.

A. FILE SECTION

The Narrator user may describe up to nineteen (19) individual input or output files within the File Section.

Each file is described in two parts. The first part provides the Narrator with information regarding the structure of the records within the file (i.e., block or message format), tape labeling requirements, and a listing of the names of individual records that appear in the file. The second part then describes each record in complete detail.

Example:

```
FILE SECTION.
FD file-name.........      (file description)
   1 record-name.....      (record descriptions)
   1 record-name.....

FD file-name.........      (file description)
   1 record-name.....      (record description)

FD file-name.........      (file description)
   1 record-name.....      (record descriptions)
   1 record-name.....

              (etc.)
```

Format for the File Description entry:

```
FD file-name        [; RECORDING MODE IS BLOCK]
[; BLOCK CONTAINS integer-1 RECORDS; RECORD CONTAINS    [integer-2 TO]
   integer-3 CHARACTERS]
                        {STANDARD                                       }
 ; LABEL RECORDS ARE    {BEGINNING-TAPE-LABEL      [, ENDING-TAPE-LABEL]}
                        {OMITTED                                        }

[; VALUE OF {ID            } IS literal]    [, ACTIVE-TIME IS literal]
            {IDENTIFICATION}

 ; DATA RECORDS ARE record-name-1    [, record-name 2, etc.......] .
```

FD identifies the beginning of a file description entry and precedes each file-name. Clauses that follow the file-name are optional in many cases. For further details, see the individual explanation for each clause.

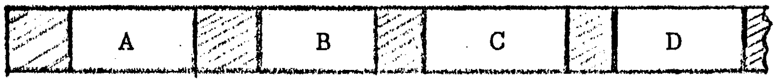
The File Description entry is terminated by a period.

RECORDING MODE

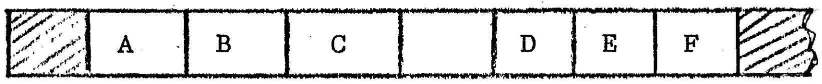
[; RECORDING MODE IS BLOCK]

Records may appear within files in one of two modes - "message" format or "block" format. "RECORDING MODE IS BLOCK" informs the Narrator that the records appear in "block" format, i.e., without start and end message symbols. For example:

Block format, (single records)

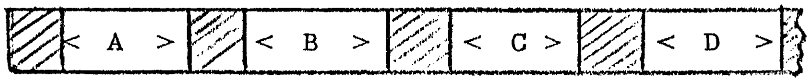


Block format, (batched records)

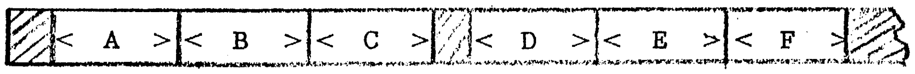


When this clause is absent the Narrator assumes that all records (singly or batched) appear in message format. For example:

Message format (single records)



Message format (batched records)



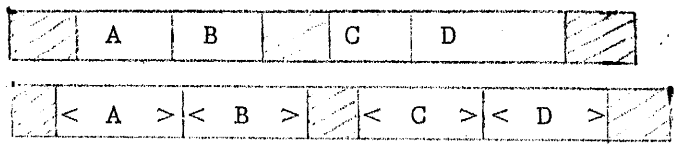
Note:

An EF symbol and three spaces has special significance for the Input/Output Control. These characters, therefore, cannot appear as the first four characters in any record that is in "block" format.

BLOCK CONTAINS

```
[; BLOCK CONTAINS integer-1 RECORDS; RECORD CONTAINS  [integer-2 TO]
   integer-3 CHARACTERS]
```

This clause is used to indicate that records in the file are grouped (batched). For example:



"Integer-1" specifies the number of records appearing in the batch. "Integer-3" provides the maximum record size. If record sizes vary within the batch, records MUST appear in message format. In this case, "integer-2" provides the minimum record size and "integer-3" provides the maximum record size.

The processing of batched records is automatically accommodated by the Input/Output Control during object program running time. For programming purposes, the Narrator user processes records as if they appeared in the files as single records. See page 13 for procedure to be followed when last "batch" does not contain the specified number of records.

Notes:

1. The maximum number of records that may appear within a "batch" is 99; the maximum size record in any batch is 9999 characters.
2. Start and End Message Symbols are not to be included in the record size.

EXAMPLES

a) ; BLOCK CONTAINS 1Ø RECORDS; RECORD CONTAINS 8Ø CHARACTERS

b) ; BLOCK CONTAINS 5 RECORDS; RECORD CONTAINS 65 TO 126 CHARACTERS

LABEL RECORDS

```
                         { STANDARD                                        }
; LABEL RECORDS ARE      { BEGINNING-TAPE-LABEL   [, ENDING-TAPE-LABEL]    }
                         { OMITTED                                         }
```

This entry specifies what type of labels are associated with the file and must be present in every File Description. (See discussion on Labels, page 126.)

As indicated in the format, a beginning label may be specified without an end label, but the reverse is not true.

Examples:

a) ; LABEL RECORDS ARE OMITTED
b) ; LABEL RECORDS ARE STANDARD
c) ; LABEL RECORDS ARE BEGINNING-TAPE-LABEL

VALUE

[; VALUE OF {ID / IDENTIFICATION} IS literal]

If the file being described contains a beginning tape label, the user must specify the value that will appear as the IDENTIFICATION (or ID) item in that label. If tape labels are omitted, this clause is not required.

When STANDARD labels are used the literal in this clause must be eight (8) characters. For example:

```
; VALUE OF ID is "MASTER++"
; VALUE OF ID is "TRANS+++"
```

For user-designed labels, the size of the literal may range from 1 to 12 characters. However, all files in the program must contain the same size ID entry. For example:

```
; VALUE OF ID IS "MAST"
; VALUE OF ID IS "TRAN"
; VALUE OF ID IS "E+++"
```

ACTIVE-TIME

[, ACTIVE-TIME IS literal]

This entry provides the Narrator with information for computing the "purge" or "obsolete" date for OUTPUT files. This clause must appear when the file being described is an OUTPUT file which has a PURGE-DATE item in its beginning tape label.

At program running time, the number of DAYS -- specified by the literal -- will be automatically added to today's date to determine the purge date for the beginning tape label. The literal may range in size from 1 to 4 characters and is surrounded by quotes.

For purposes of computing the purge date, January has 31 days, February has 28, March has 31, April has 30, etc. Thus, if the active time is "5", the purge date computed by the program on May 30, 1961, would be 060461; on June 30, 1961, the computed date would be 070561.

Examples:

```
; ACTIVE TIME IS "6Ø"
; ACTIVE TIME IS "4"
```

<u>DATA</u> <u>RECORDS</u>

; <u>DATA</u> <u>RECORDS</u> ARE record-name-1 [, record-name-2, etc....] .

This clause is required in all File Descriptions and must appear as the last information in the File Description. It is used by the Narrator to cross-reference the data records associated with the file.

The presence of more than one record-name indicates that the file contains more than one type of data record. These records may vary in size, have different formats, and need bear little relationship to one another. Similarly, the order in which they are listed is not significant.

When a file contains more than one type of data record, it must be remembered that all records within the file will share the same computer memory area. Thus, if the TRANSACTION-FILE contains both RECEIPTS and DISBURSEMENTS records, a read command will bring the next "logical" record into the Transaction-File area. It must be determined through programming, however, whether this is a RECEIPTS record or a DISBURSEMENTS record. (When the record type has been established, the programmer may then refer to data fields within that record by their appropriate names.)

Examples:

a) ; DATA RECORDS ARE TRANSACTIONS.

b) ; DATA RECORDS ARE RECEIPTS, DISBURSEMENTS, CHANGES.

NOTES:

When the first file includes record entries defining tape labels, these entries are not considered as data records, and their names must not appear in this clause.

A maximum of twenty-six (26) different type Records may be described for each file within the program.

RECORD DESCRIPTION

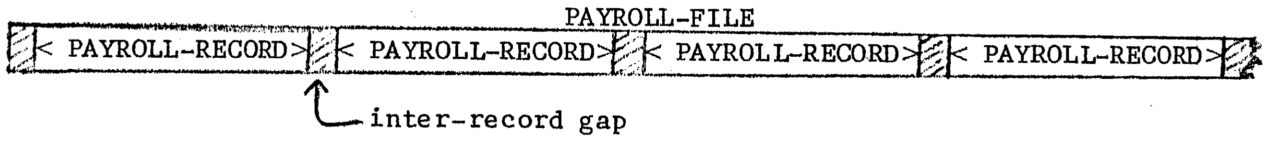
Introduction

"Record" is a generic name that identifies a logical group of information units pertaining to the same subject, area, individual, etc. For example, we can specify "PAYROLL-RECORD" as the generic name that identifies the following units of information:

```
                 ( EMPLOYEE-NAME
                 | DEPARTMENT
PAYROLL-RECORD  <  HOURS-WORKED
                 | HOURLY-RATE
                 ( NET-PAY
```

Thus, when we specify "PAYROLL-RECORD" we are not referring to a single unit of information but, more accurately, to a group of units (often called "items") that make up the PAYROLL-RECORD.

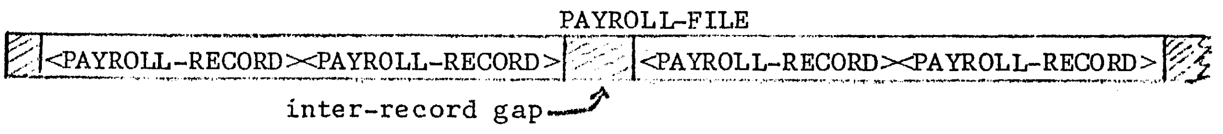
For data processing purposes, each PAYROLL-RECORD can be considered as a "logical" record within the PAYROLL-FILE, and in the case of magnetic tape, would appear as follows:



The programmer, then, may process this file in the following manner:

1. READ PAYROLL-FILE. -- bring next "logical" record from magnetic tape into the computer.
2. MULTIPLY HOURLY-RATE BY HOURS GIVING GROSS-PAY. } compute employee's net pay for this record.
3. SUBTRACT DEDUCTIONS FROM GROSS-PAY. }
4. WRITE PAYROLL-RECORD. -- write out updated record to output file.
5. GO TO 1. -- bring in next record and repeat cycle.

Generally, each "logical" record within the file is carried as a "physical" record on tape, i.e., preceded and followed by an inter-record gap. Thus, when the programmer requests a record from the file, one physical record (or one logical record) is brought into the computer. Another method, however, is to group a number of "logical" records into one "physical" record; this is called "batching."


PAYROLL-FILE
<PAYROLL-RECORD><PAYROLL-RECORD>
<PAYROLL-RECORD><PAYROLL-RECORD>
inter-record gap

In this case, the programmer has grouped or batched his records. Thus, when the computer executes a read command for this file, a number of "logical" records will be entered, instead of only one. (When record "batching" is employed, the Narrator Input/Output Control will place one "logical" record into the file area for each READ command - see page 108)

Describing the Organization of Records

Individual information units within a record must, of course, be organized and assigned a specific sequence in order to properly locate and process these items. This organization, along with a specific description of each field, must be provided by individual data record entries.

The Narrator user may organize records in either a "fixed-field" format, a "variable" format, or a combination of both. The fixed-field method is no doubt familiar to most of us. In this system, a unit of information, such as employee-name, is assigned a definite area within the record -- columns 10 through 35, for instance, in a punched card. Another method -- one used to great advantage in the 501 system -- is to organize data in sequence, but not in fixed-field format. This treats variable data in its natural form and obviates carrying redundant or extra characters for positioning. This accomplished through the use of the Item Separator, SM and EM symbols, which are explained in other 501 publications.

FORMAT FOR A DATA RECORD ENTRY

Option 1:

```
Level-number   data-name ; SIZE IS [integer-1 TO] integer-2 CHARACTERS

[ ; CLASS IS  { AN           } ]
[             { ALPHABETIC   } ]
[             { ALPHANUMERIC } ]
[             { NUMERIC      } ]

[; SIGNED]  [; POINT LOCATION IS LEFT integer-3 PLACES]

[ ; JUSTIFIED { LEFT  } ]
[             { RIGHT } ]

[ ;  { VALUE IS literal                                            } ]
[    { PICTURE IS (any combination of allowable characters)        } ]
[                            maximum of 18                            ]

[ ;  { ZERO SUPPRESS     }                                  ]
[    { CHECK PROTECT     }  [LEAVING integer-4 PLACES]      ]
[    { FLOAT DOLLAR SIGN }                                  ]

[; BLANK WHEN ZERO]

[; OCCURS integer-5 TIMES] .
```

Option 2:

```
Level-number   record-name-1; COPY   record-name-2.
```

Level-number

A 501 Narrator Record Description consists of a number of entries, each entry defining the characteristics of a particular unit of data within the record. To differentiate between records and units of in-information that appear within records, the concept of "level number" has been introduced. There are five level numbers:

Level No.	Function
1	defines the largest element of data, i.e., a logical record.
2	distinguishes a particular unit of information within the record, i.e., an item.
3	denotes that this information is part of an item, i.e., a sub-item.
77	a special level number assigned to non-contiguous constants and working-storage areas -- see Constant and Working-Storage Sections.
88	specifies a "condition-name" that the user assigns to a value that may appear in the preceding level 2, 3 or 77 entry

To clarify this concept of level numbers, let us take a hypothetical file that contains two types of records, organized as follows:

SHIPPING-ORDER RECORD

< •Code•Date•Stock-Number•Quantity•Unit-Price•Amount >

BACK-ORDER RECORD

< •Code•Warehouse•Stock-Number•Quantity >

In defining these records by Record Description entries, the following hierarchical structure is followed:

```
1 SHIPPING-ORDER                  1 BACK-ORDER
  2 CODE                            2 CODE
  2 DATE                            2 WAREHOUSE
    3 MONTH                         2 STOCK-NUMBER
    3 DAY                           2 QUANTITY
    3 YEAR
  2 STOCK-NUMBER
  2 QUANTITY
  2 UNIT-PRICE
  2 AMOUNT
```

This example, of course, does not show the complete entries, but only the relationship of level-numbers. As indicated, level-1 entries pertain to the overall record; level-2 entries pertain to the items that appear within a record; and level-3 entries provide a further breakdown of level-2 entries.

Sub-Items (Level-3)

Sub-items are designated by the level number "3." For example, assume that a bank assigns an 8-character customer account number in which the first two characters signify the branch office, and the third character identifies the type of account. This item may be sub-divided in the record entry as follows:

```
2 ACCOUNT-NUMBER; SIZE IS 8.
  3 BRANCH; SIZE IS 2.
  3 TYPE; SIZE IS 1.
  3 FILLER; SIZE IS 5.
```

When an item is sub-divided the entire item must be broken down into sub-items. If sub-item addressing is not required for part of the field, the data name "filler" should be used. (Note that FILLER cannot be used as a data-name within a procedural statement- see note 2 under Data-name, page 30.)

Condition-Names (Level 88)

Let us assume that the item, warehouse, contains one character, and that the warehouse location depends on which character (i.e., value) is present. For example: if an "A" appears, the Boston warehouse is indicated; "B" pertains to the New York warehouse, and so on. This item then is considered as a "conditional-variable" -- one which may assume any condition that the user names and defines.

Example:

```
1 BACK-ORDER; SIZE IS 15.
  2 WAREHOUSE; SIZE IS 1.
    88 BOSTON, VALUE IS "A".
    88 NEW-YORK, VALUE IS "B".
    88 DALLAS, VALUE IS "C".
    88 LOS-ANGELES, VALUE IS "D".
  2 STOCK-NUMBER; SIZE IS 8.
  2 QUANTITY; SIZE IS 6.
```

In this example the Narrator is told that the value in the record for the condition-name DALLAS is "C." In the procedural division the programmer writes the following statement: IF DALLAS GO TO 116. Because DALLAS is a condition-name, the Narrator will look for its value (in this case,"C") and will then compare this value to the Warehouse item in the Back-Order record. If the Warehouse item contains a "C" the program will transfer to line 116; if not, the next statement will be performed.

To indicate that we are defining a condition-name (i.e., qualifying the preceding information and not introducing new data) the following format is used for condition-names:

a) level-number -- must be 88.

b) data-name -- follows usual data-name rules (see DATA-NAME)

(cont'd)

c) VALUE IS literal -- the exact "value" that will appear in the record for this condition-name is specified by the "literal." Literals used in this clause may not exceed 16 characters and asterisks or slant symbols may not be used.

   When describing the value, the programmer ignores the ISS symbol (if present) and only describes the data appearing in the field. (See VALUE, page 37)

d) Condition-names may only be used within IF statements in the PROCEDURE DIVISION.

Only this information (i.e., LEVEL, NAME, VALUE) can appear for these condition-name entries.

Data-name

The Narrator user may assign data-names (up to 30 characters, including hyphens) to identify a file, record, item, sub-item, condition, working storage area or constant.

Data-names must be unique for all file-names, record-names (level-1), and level-77 data fields. Data-names appearing within a record (level-2, level-3 and level-88) need only be unique within the same record description. For example:

```
1 MASTER-RECORD          1 TRANSACTION-RECORD
  2 CUSTOMER-ACCT          2 CUSTOMER-ACCT
  2 ADDRESS                2 ADDRESS
  2 CITY-STATE             2 CITY-STATE
  2 AMOUNT                 2 AMOUNT
  2 DATE                   2 DATE
    3 MONTH                  3 MONTH
    3 DAY                    3 DAY
    3 YEAR                   3 YEAR
```

Qualifying data-names

When the user wishes to refer to a non-unique data-name (level 2, 3 or 88), he must qualify that name by the record in which it appears. This is accomplished by following the data-name by the word OF or IN, and the name of the record in which it appears. For example:

```
MOVE AMOUNT OF MASTER-RECORD TO BALANCE.

MOVE AMOUNT OF TRANSACTION-RECORD TO BALANCE.

IF MONTH IN MASTER-RECORD EQUALS
   MONTH IN TRANSACTION RECORD GO TO 41Ø.
```

NOTES

1. Records may contain up to 999 data fields; this includes all level-2 and level-3 entries, but does not include condition-name entries (level-88).

2. The word "FILLER" may be used as a data-name in a record description to define areas whose contents are not referenced within the program. Because of this special use, a FILLER area cannot be addressed by any procedural statement.

   Example:

   ```
   1 INVENTORY; SIZE IS 57; CLASS NUMERIC.
     2 STOCK-NUMBER; SIZE IS 7.
     2 FILLER; SIZE IS 1Ø.
     2 QUANTITY-ON-HAND; SIZE 1Ø.
     2 FILLER; SIZE IS 3Ø.
   ```

3. The use of the COPY option (see page 51) always produces non-unique data-names.

SIZE

; SIZE IS [integer-1 TO] integer-2 CHARACTERS.

All record entries, except condition-names (level-88), must contain a SIZE clause. (If the data-name varies in size, or if the data-name does not always appear in the record, the integer-1 TO integer-2 notation must be used.)

Level-1 entries:

The maximum size of a level-1 entry (complete record) is 9999 characters. Record size is determined by totaling the maximum size of all level-2 entries within the record, but NOT including start and end message symbols.

Level-2 entries:

The maximum size of a level-2 entry (item) is 999 characters. The size MUST include the ISS, if present. (The total of the level-2 entries must of course agree with the maximum size of the level-1 entry with which they are associated.)

Level-3 entries:

The size of a level-3 entry must a) be fixed, and b) must NOT include an ISS. Therefore, if the item contains an ISS symbol, the total of the level-3 entries will be one less than the item size. If an ISS is not associated with the item, the total of the level-3 entries will agree with the size of the item. For example:

```
(with ISS)                       (without ISS)

2 DATE; SIZE 7.                  2 DATE; SIZE 6.
  3 MONTH; SIZE 2.                 3 MONTH; SIZE 2.
  3 DAY; SIZE 2.                   3 DAY; SIZE 2.
  3 YEAR; SIZE 2.                  3 YEAR; SIZE 2.
```

If a level-3 entry is associated with a level-2 entry that is non-FAA (i.e., not fixed or always appearing), the level-3 entry cannot be referenced in the variable area. (The item, however, may be moved to a fixed area and level-3 items may then be directly addressed.)

Level-88 entries:

Level-88 (condition-names) entries do not include SIZE clauses since the size is determined from the entry with which it is associated.

EXAMPLE:

```
1 REQUISITIONS; SIZE 19 TO 23; CLASS NUMERIC.
  2 STOCK-NUMBER; SIZE 8.
  2 DATE; SIZE 7.
    3 MONTH; SIZE 2.
    3 DAY; SIZE 2.
    3 YEAR; SIZE 2.
  2 CODE; SIZE 2.
    88 RECEIPT; VALUE IS "R".
    88 WITHDRAWAL; VALUE IS "W".
  2 QUANTITY; SIZE 2 TO 6.
```

When describing the value, the programmer ignores the ISS symbol (if present) and only describes the data that will appear in the field for that condition-name.

$\boxed{\underline{\text{CLASS}}}$

$$\left[\ ;\ \underline{\text{CLASS}}\ \text{IS}\ \left\{\begin{array}{l}\underline{\text{AN}}\\ \underline{\text{ALPHABETIC}}\\ \underline{\text{NUMERIC}}\\ \underline{\text{ALPHANUMERIC}}\end{array}\right\}\ \right]$$

This clause defines the type of data that will appear in the field being described. Note that AN is an accepted abbreviation for ALPHANUMERIC.

The CLASS of a data field is determined in the following manner:

1. NUMERIC - field contains the digits 0-9 (with or without sign)

   Examples: •12456-
   •4287

2. ALPHABETIC - field does <u>NOT</u> contain a numeric character.

   Examples: •JOHN-JONES---
   •NEW-YORK-CITY,-N.Y.
   •DESK-MODEL(A)

3. ALPHANUMERIC - a) field contains alphabetic and numeric characters

   Examples: •4Ø872A

   •APRIL-25-1965

   b) field contains numerics and editing symbols (See exception below)

   Examples: •$18.75
   •11-15-61

   Exception: If the user is describing a field that will <u>receive</u> numeric data, AND EDITING IS TO BE <u>PERFORMED</u> ON THAT FIELD, the field should be defined as NUMERIC. ("Editing" includes zero-suppression, check protect, float dollar sign, blank when zero, and insertion of decimal points, dollar signs and commas.)

A CLASS clause must appear for every level-1 entry; other levels may or may not require a class clause as explained below:

<u>Level-1 defined as NUMERIC</u>

When a level-1 entry has been defined as NUMERIC all data entries in that record are automatically classed as numeric, and cannot be redefined otherwise.

Example:

```
1 MASTER; SIZE 27; CLASS NUMERIC.
  2 ACCOUNT-NO; SIZE 14.
  2 AREA; SIZE 3.
  2 AMOUNT; SIZE 1Ø.
```

Level-1 defined as ALPHABETIC

When the level-1 entry has been defined as ALPHABETIC all data entries within that record are automatically classed as alphabetic, and cannot be redefined otherwise.

Example:

```
1 CUSTOMER-RECORD; SIZE 21 TO 47; CLASS ALPHABETIC.
  2 CODE; SIZE 3.
  2 NAME; SIZE 1Ø TO 3Ø.
  2 CITY; SIZE 8 TO 14.
```

Level-1 defined as ALPHANUMERIC

When a level-1 entry is defined as ALPHANUMERIC, any or all of the associated level-2 and level-3 entries may be specifically designated as being numeric or alphabetic. Note, however, that once a class has been assigned, all succeeding entries will be assigned the same class until another class is specified.

For example:

```
1 PAYROLL-RECORD; SIZE 41 TO 68; CLASS AN.
  2 EMPLOYEE-NO; SIZE 6; CLASS NUMERIC.
  2 DIVISION; SIZE 2.
  2 HOURS-WORKED; SIZE 3.
  2 HOURLY-RATE; SIZE 4.
  2 NAME; SIZE 1Ø TO 25; CLASS AN.
  2 ADDRESS; SIZE 8 TO 16.
  2 CITY-STATE; SIZE 8 TO 12.
```

In this example DIVISION, HOURS-WORKED and HOURLY-RATE will be classified as numeric, since the last specific class entry was numeric. When "alphanumeric" was assigned to NAME, the succeeding items (ADDRESS and CITY-STATE) were classified as alphanumeric also.

Level-3 entries

Classes assigned to level-3 entries cannot contradict the class for the level-2 entry with which they are associated. For example, the following is INCORRECT:

```
2 ACCOUNT-NO; SIZE 5; CLASS NUMERIC.
  3 BRANCH; SIZE 2.
  3 OFFICE; SIZE 2; CLASS ALPHABETIC.
```

NOTE

During compilation, the classes of all operands involved in arithmetic operations are verified for class compatability. If the classes are opposed (i.e., if all fields are not numeric) an error will be indicated. Similarly, an error print-out will also occur if a comparison or move operation is attempted which involves fields that are opposite in nature. It would be illegal, for example, to move or compare numeric fields to alphabetic fields. (See rules governing ADD, IF and MOVE verbs.)

SIGNED

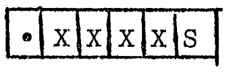
[; SIGNED]

The SIGNED clause is only used in conjunction with numeric data. This clause specifies that the numeric data being described contains a sign located in the least significant position of the field (RHE).

When signs are present they must be reflected as a character location in the SIZE clause. In addition, if the field structure is further described by a PICTURE clause, the sign position must also be reflected in the PICTURE.

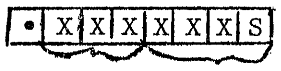
A numeric field containing four characters, a sign location, and an ISS would be defined as follows:

```
2 AMOUNT; SIZE IS 6; CLASS NUMERIC; SIGNED.
```



Note that the sign appears as the least significant character within the field. If an item (level-2) is broken down into sub-items (level-3), the sign location is a character position within the last sub-item described. For example:

```
2 TOTAL-EXPENSE; SIZE 8; CLASS NUMERIC; SIGNED
  3 THOUSANDS; SIZE 3.
  3 HUNDREDS; SIZE 4.
```



The Total-Expense field could also be described as follows:

```
2 TOTAL-EXPENSE; SIZE 8; CLASS NUMERIC; SIGNED.
  3 THOUSANDS; SIZE 3.
  3 HUNDREDS; SIZE 3.
  3 FILLER; SIZE 1.
```

POINT LOCATION

[; POINT LOCATION IS LEFT integer PLACES]

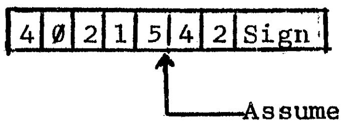
This clause specifies the position of the assumed decimal point (to the left of the least significant digit) for numeric data. An actual decimal point may not be defined by this clause; this is accomplished by the PICTURE.

"Integer" is one numeric character, ranging from 1 to 9; the value "Ø" cannot be used.

Examples:

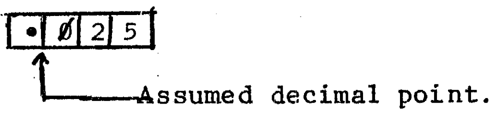
2 BALANCE; SIZE 8; CLASS NUMERIC; SIGNED; POINT LEFT 2.

BALANCE FIELD



2 STATE-TAX; SIZE 4; CLASS NUMERIC; POINT LEFT 3.

STATE-TAX FIELD



NOTE:
The POINT LOCATION clause can only appear for NUMERIC fields.

VALUE

```
[ ; VALUE IS literal ]
```

The "VALUE" clause is used to describe the exact "value" that will appear for a condition-name (level-88). This clause is not used with level-1, 2 or 3 entries.

The "literal" used in this clause may not exceed 16 alphanumeric characters; a quote symbol, slant or asterisk cannot appear as one of the 16 characters.

Example A:

```
2 TRANSACTION-CODE; SIZE 3.
     88 WITHDRAWAL; VALUE IS "14".
     88 DEPOSIT; VALUE IS "26".
```

(Note that when describing the value, ISS symbols are ignored, and only the actual data is shown in the literal.)

Example B:

```
2 CUSTOMER-CODE; SIZE 8; CLASS NUMERIC.
  3 FILLER; SIZE 5.
  3 DISTRICT-OFFICE; SIZE 2.
    88 CHICAGO; VALUE IS "42".
    88 ATLANTA; VALUE IS "35".
    88 BOSTON; VALUE IS "62".
    88 NEW-YORK; VALUE IS "18".
    88 LOS-ANGELES; VALUE IS "22".
    88 DALLAS; VALUE IS "17".
```

JUSTIFIED

```
[ ; JUSTIFIED {LEFT } ]
[             {RIGHT} ]
```

This option is only required when STANDARD rules of positioning are NOT desired.

"Justification" is necessary whenever the "net" size of a sending field is less than the "net"size of a receiving area. This situation arises in the following cases:

a) moving a variable field to a fixed field

b) moving a smaller fixed field to a larger fixed field

c) when a field receives the result of an arithmetic operation.

"Justification" simply means the manner in which data will be positioned in a receiving field -- that is, will the data to be placed into the area starting at the left-hand end, or at the right-hand end. If justification is necessary, the remainder of the area will be "padded," i.e., filled with zeros or spaces.

Standard Justification

The "standard" rules for positioning within a field are:

A. If the "receiving" field is NUMERIC, the data will be right-justified with zero-fill. For example:

•	Ø	Ø	Ø	Ø	1	4	6

•	Ø	4	5	8	1	2	2

B. If the "receiving" field is ALPHABETIC or ALPHANUMERIC, data will be left-justified, with space-fill. For example:

•	J	O	E	-	J	O	N	E	S	-	-	-	-	-	-

•	H	A	R	R	Y	-	J	O	H	N	S	O	N	-	-

Non-Standard Justification

If JUSTIFIED RIGHT is specified for an ALPHABETIC or AN receiving field, data is placed into this field with space-fill to the left. For example:

•	-	-	-	-	-	-	J	O	E	-	J	O	N	E	S

When JUSTIFIED LEFT is specified for a NUMERIC receiving field, data is placed into this field with zero-fill to the right. For example:

•	1	4	6	Ø	Ø	Ø	Ø

Note that when data is moved to a numeric field that is specified as "JUSTIFIED LEFT," the point location in the receiving field will be ignored, and the data will be placed into the receiving area beginning at the left-hand end.

NOTES:

1. When "standard" justification is desired the JUSTIFIED clause is not required.

2. Justification is performed when data is moved into an area, or when the area receives the result of an arithmetic operation.

3. Note that the type of justification will always be determined by the "receiving" field description. Thus, depending on the "contents" of the sending field, illogical results may occur. For example:

Sending (ALPHANUMERIC) Receiving (NUMERIC)

$$\boxed{\bullet}\boxed{J}\boxed{O}\boxed{N}\boxed{E}\boxed{S} \longrightarrow \boxed{\bullet}\boxed{\emptyset}\boxed{\emptyset}\boxed{\emptyset}\boxed{\emptyset}\boxed{J}\boxed{O}\boxed{N}\boxed{E}\boxed{S}$$

Sending (ALPHANUMERIC) Receiving (ALPHABETIC)

$$\boxed{\bullet}\boxed{1}\boxed{2}\boxed{3}\boxed{4}\boxed{5} \longrightarrow \boxed{\bullet}\boxed{1}\boxed{2}\boxed{3}\boxed{4}\boxed{5}\boxed{-}\boxed{-}\boxed{-}\boxed{-}\boxed{-}$$

Sending (NUMERIC "edited") Receiving (NUMERIC)

$$\boxed{\bullet}\boxed{\$}\boxed{5}\boxed{.}\boxed{8}\boxed{8} \longrightarrow \boxed{\bullet}\boxed{\emptyset}\boxed{\emptyset}\boxed{\emptyset}\boxed{\$}\boxed{5}\boxed{.}\boxed{8}\boxed{8}$$

Sending (NUMERIC "edited") Receiving (NUMERIC)

$$\boxed{-}\boxed{-}\boxed{-}\boxed{4}\boxed{2}\boxed{5} \longrightarrow \boxed{\emptyset}\boxed{\emptyset}\boxed{\emptyset}\boxed{\emptyset}\boxed{-}\boxed{-}\boxed{-}\boxed{4}\boxed{2}\boxed{5}$$

PICTURE

```
; PICTURE IS (any combination of allowable characters)
                        maximum of 18
```

This clause can only be used with a level-2 or level-3 entry, or with a level-77 working storage entry.

The PICTURE clause has two functions:

1) To provide a pictorial description of ALPHABETIC or ALPHANUMERIC fields. For example:

   PICTURE IS KAAAABBAA
   PICTURE IS K999AA

   When used with ALPHABETIC or ALPHANUMERIC fields, then, the PICTURE clause is for programmer reference only; it does not create coding in the object program, nor does it bring about any modification or examination of data moved into the field.

2) To specify editing functions that are to be performed on a NUMERIC field when numeric data is moved into that field, or when that field receives the result of an arithmetic operation. For example:

   PICTURE IS K$99,999.99S
   PICTURE IS K$9,999BB99

   Unlike the first function, the use of the PICTURE clause with NUMERIC fields does produce object-program coding. That is, dollar signs, commas, decimal points, blanks, etc. are physically inserted into the field when numeric data is moved into it.

The maximum number of characters that may appear in the PICTURE clause is 18. The allowable characters and their meanings are as follows:

Character	Significance
9	a numeric character.
Ø	a position into which a zero is to be inserted.
B	a position into which a space is to be inserted.
S	a position into which a positive or negative sign is to be stored.
$	a position where a dollar sign is to be inserted. (see EDITING).
.	a position where an actual decimal point is to be inserted. (see EDITING). Note that a decimal point cannot be the last character in the PICTURE.
,	a position where a comma is to be inserted. (see EDITING).
A	an alphabetic character.
X	an alphanumeric character.
J	the first position in the field (LHE) is NOT an Item Separator (ISS).
K	the first position in the field (LHE) is an Item Separator. (When an ISS appears in the field, this position must also be reflected in the SIZE clause.)

AUTOMATIC EDITING OF NUMERIC FIELDS

A numeric field is considered to be automatically "edited" when the user desires the insertion of one or more of the following characters:

$	dollar sign
,	comma
.	decimal point
B	blank (space)
Ø	zero

When these characters are desired, the user must describe the field by a PICTURE clause. For example:

```
SIZE IS 6; PICTURE IS K99B99
SIZE IS 1Ø; PICTURE IS K$9,999.99
SIZE IS 11; PICTURE IS K$9,999.ØØS
SIZE IS 9; PICTURE IS J$9,999.99
```

Note that the SIZE clause must include the total number of characters in the field, and that the presence (K) or absence (J) of an ISS must be indicated by the first character.

RECORDS THAT APPEAR IN MESSAGE FORMAT

When the user describes a record as being in MESSAGE FORMAT, the Narrator will assume that ALL items (level-2) within that record contain an ISS symbol. Thus, if the size of an item is 8, it will be assumed that the field contains an ISS and seven characters of data.

If a level-2 entry within the message does NOT contain an ISS, the programmer must make this known to the Narrator by specifying that:

PICTURE IS J.

Therefore, if the item size is 8, and PICTURE IS J is specified, it will be assumed that the field contains eight characters of data.

When a PICTURE is used with a level-3 entry, the first character in the PICTURE should always be "J," because level-3 entries cannot contain ISS symbols.

RECORDS THAT APPEAR IN BLOCK FORMAT

When a record appears in block format, ISS symbols are usually not present; and the Narrator will assume that ALL items (level-2) within that record appear without ISS symbols. Therefore, size is 8 would denote that the field contains eight characters of data.

If any item contains an ISS symbol, the user must indicate this by specifying: PICTURE IS K. Thus, SIZE 8; PICTURE IS K would specify that this field contains an ISS and seven characters of data.

Alternate Method For Expressing Characters Within Field

When using the PICTURE clause, the programmer may indicate by a decimal number -- enclosed by parenthesis -- that the preceding character appears in the field "n" times. For example, if the field contains six numerics followed by two alphabetics, any of the following examples would be permissable:

a) PICTURE IS J999999AA
b) PICTURE IS J9(6)AA
c) PICTURE IS J9(6)A(2)

Since a maximum of 18 characters may only appear in the PICTURE clause, this option is especially useful when the field contains more than eighteen character locations. For example:

PICTURE IS JA(14)999999BB

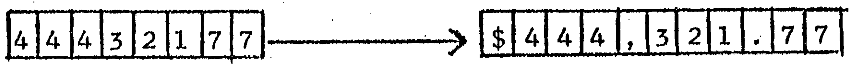
GENERAL COMMENTS REGARDING USE OF PICTURE CLAUSE

1. PICTURE clauses can only be used with level-2 or level-3 entries, or with a level-77 working storage area.

2. When automatic editing is desired, the receiving field must be NUMERIC and the data in the sending area is considered to be NUMERIC without editing symbols.

Example:

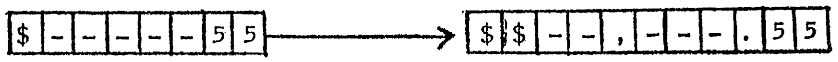
Sending (AN) Receiving (NUMERIC; PICTURE IS J$999,999.99)



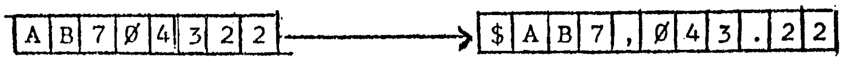
If editing symbols appear within the sending field, editing will be attempted on the data appearing in the field and the result will be unpredictable.

Example:

Sending NUMERIC (EDITED) Receiving (NUMERIC; PICTURE IS J$999,999.99)



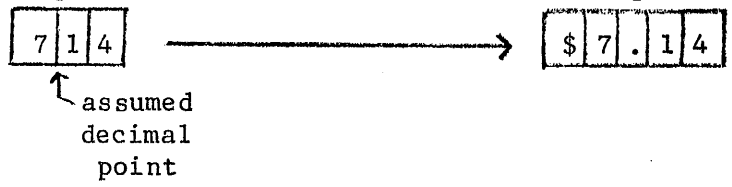
Sending (AN) Receiving (NUMERIC; PICTURE IS J$999,999.99)



3. When the PICTURE contains a decimal point, a POINT LOCATION clause must also appear for that entry. The position of the decimal point should be the same in both clauses:

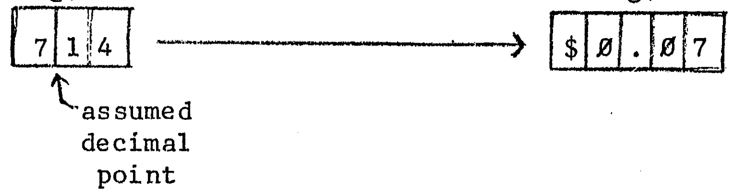
<u>CORRECT</u>

Sending; POINT LEFT 2 Receiving; POINT LEFT 2; PICTURE J$9.99



<u>INCORRECT</u>

Sending; POINT LEFT 2 Receiving; PICTURE J$9.99



Note: In this example, the receiving field did not have a POINT LOCATION clause, thus, the move operation considered the receiving field as consisting of a whole number without decimal locations. ("Editing", then, was applied to the integral digits transferred.)

4. When the PICTURE contains a "sign" (S) the SIGNED clause must also appear for that entry.

5. The SIZE clause must reflect all locations shown in the PICTURE clause. For example:

```
2 AMOUNT; SIZE 11; PICTURE IS J$99,999B99S.

2 BALANCE; SIZE 12; PICTURE IS K$99,999B99S.

2 BALANCE; SIZE 9; PICTURE IS J9(6).99.
```

EDITING

Function: To specify special editing operations that the user wishes performed on a field when data is moved into it, or when the field receives the result of a decimal operation. (Note that the editing function can only be specified for NUMERIC fields.)

```
[  ⎧ZERO SUPPRESS     ⎫                              ]
[ ;⎨CHECK PROTECT     ⎬ [LEAVING integer   PLACES]   ]  [; BLANK WHEN ZERO]
[  ⎩FLOAT DOLLAR SIGN ⎭                              ]
```

ZERO SUPPRESS

When ZERO SUPPRESS is specified, all leading "zeros" up to the decimal point (real or assumed) will be replaced by spaces. If the LEAVING option is included, the designated number of integer places (to the left of the decimal point) will not be checked for zeros.

Examples:

```
     Sending Field              Receiving Field
A.   |Ø|Ø|Ø|Ø|Ø|6| ——————→ |Ø|Ø|Ø|Ø|Ø|6|

     Sending Field              Receiving Field (ZERO SUPPRESS)
B.   |Ø|Ø|Ø|Ø|Ø|6| ——————→ |-|-|-|-|-|6|

     Sending Field              Receiving Field (ZERO SUPPRESS LEAVING 2 PLACES)
C.   |Ø|Ø|Ø|Ø|Ø|6| ——————→ |-|-|-|-|Ø|6|


     Sending Field (POINT LEFT 2)   Receiving Field (POINT LEFT 2; ZERO SUPPRESS
D.                                                    LEAVING 2 PLACES)

     |Ø|Ø|Ø|Ø|Ø|6|  ——————→  |-|-|Ø|Ø|Ø|6|
              ↑                      ↑
              └Assumed               └Assumed
               decimal                decimal
               point                  point
```

CHECK PROTECT

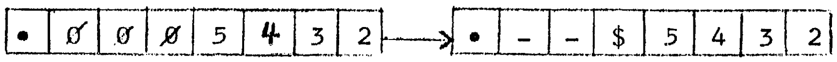
This option is identical to ZERO SUPPRESS except that leading zeros will be replaced by "asterisks" instead of "spaces". For example:

```
     Sending Field              Receiving Field (CHECK PROTECT)

     |•|Ø|Ø|Ø|5|Ø|3|2| ——→ |•|*|*|*|5|Ø|3|2|

     Sending Field              Receiving Field (PICTURE IS K$9999999 CHECK PROTECT)
     |•|Ø|Ø|Ø|5|Ø|3|2| ——→ |•|$|*|*|*|5|Ø|3|2|
```

FLOAT DOLLAR SIGN

This option is also identical to ZERO SUPPRESS, except that a dollar sign will be inserted into the location of the last leading zero. For example:

Sending Field Receiving Field (FLOAT DOLLAR SIGN)

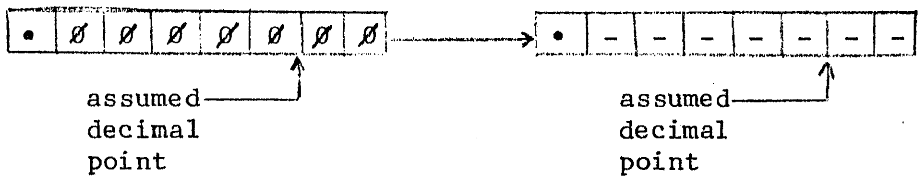


Note: In order to allow for a dollar sign insertion, at least one leading zero must exist in the data. If it does not, the results will be unpredictable.

BLANK WHEN ZERO

When this option is used, and the value of the field is "zero," all other editing options are overruled and the field is converted to spaces.

Sending Field (POINT LEFT 2) Receiving Field (POINT LEFT 2; FLOAT DOLLAR SIGN; BLANK WHEN ZERO)

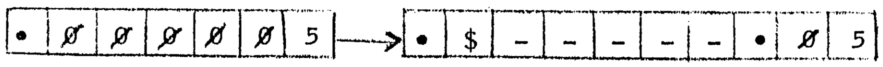


IMPORTANT NOTES

1. USING EDITING OPTIONS WITH PICTURE CLAUSE

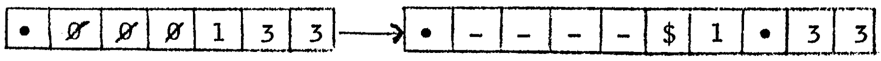
a) When ZERO SUPPRESS and CHECK PROTECT are used in conjunction with PICTURE, redundant commas will not be inserted in the field.

Sending Field Receiving Field (PICTURE IS K$9,999.99; ZERO SUPPRESS)



b) When FLOAT DOLLAR sign is used with a PICTURE clause, the dollar sign may not necessarily appear in the position shown in the PICTURE.

Sending Field Receiving Field (PICTURE IS K$9,999.99; FLOAT DOLLAR SIGN)



OCCURS

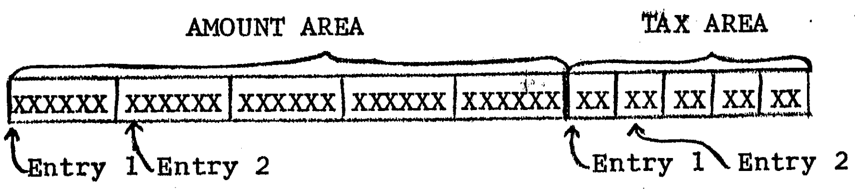
; OCCURS integer TIMES

The OCCURS clause enables the programmer to create areas within a record for tables or other homogeneous sets of data. This option may only be used with level-2 entries.

Assume that the programmer wants to define a record consisting of five amount fields, followed by five tax fields. Instead of describing this record as containing ten individual data fields, it may be defined as consisting of two tables. For example:

```
1 TABLE; SIZE 40; CLASS NUMERIC.
  2 AMOUNT; SIZE 6; SIGNED; OCCURS 5 TIMES.
  2 TAX; SIZE 2; PICTURE IS J; OCCURS 5 TIMES.
```

The file area for this record would appear as follows:

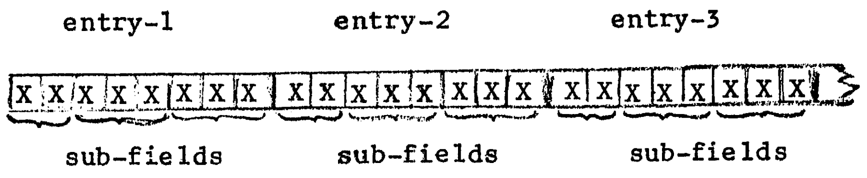


As indicated, five 6-character fields (entries) will be reserved for the amount area, and five 2-character fields (entries) will be reserved for the tax area. The format for each amount entry will be identical, i.e., $\boxed{\bullet}\boxed{X}\boxed{X}\boxed{X}\boxed{X}\boxed{S}$. Each tax entry will also be identical, i.e., two numeric characters.

If desired, a level-2 entry containing an OCCURS clause may be further described by sub-items (level-3). For example:

```
2 TAX-TABLE; SIZE 8; CLASS NUMERIC; PICTURE IS J; OCCURS 10 TIMES.
  3 STATE-CODE; SIZE 2.
  3 TAX-RATE-STATE; SIZE 3.
  3 TAX-RATE-FEDERAL; SIZE 3.
```

Because of this entry, an 80-character area will be reserved for the tax-table; this area will consist of ten 8-character entries, each of which will contain three sub fields:



Note that each entry is considered as being a numeric field of 8 characters (without an ISS).

Rules Governing the Use of the OCCURS Option.

a) The OCCURS clause can only appear in a level-2 entry.

b) The size of each entry within the table must be fixed.

c) "Integer" must be a decimal number ranging from 1 to 999, and represent the total number of occurrences.

d) Care should be exercised when computing the maximum record size for a level-1 entry when level-2 entries contain the OCCURS clause. In these instances, the size of the level-2 entry is actually the item size multiplied by the number of occurrences.

e) If an ISS or sign is indicated in the level-2 description, each entry in the table will be assumed by the Narrator to have an ISS and sign within the field.

f) Data for tables is not generated by the Narrator. Loading tables must be accomplished by the user by one of the following methods:

   1) Loading into a level-1 working storage area through use of the ACCEPT verb.

   2) Loading the table through individual MOVE's of literals. It is recommended that this be accomplished in a "housekeeping" section which can be overlaid by subsequent sections.

   3) Assigning a file area to the table and loading it from a peripheral device.

Addressing Tables

When a table (or list) has been created through an OCCURS clause, the first table entry may be directly addressed by its data-name. ALL OTHER ENTRIES MUST BE ADDRESSED BY SUBSCRIPTING.

SUBSCRIPTING

"Subscripting" provides a means by which the user may address entries in a table or list that has been created by an OCCURS clause.

There are two methods of subscripting. The first, subscripting by decimal number, is used when the location of the specific entry in the table is known to the programmer. The second, subscripting by a data field, is used when the location of the entry is not known until object program running time, or when the programmer wishes to vary a subscript reference in order to address another table entry.

Subscripting is accomplished by following the data-name with a decimal number or a data field enclosed by parenthesis. For example:

```
AMOUNT (5)
AMOUNT (CODE)
```

The data-name being subscripted must be the name of a level-2 entry that contains an OCCURS clause, or one of the level-3 data-names associated with that level-2 entry.

When the data-name being subscripted is not unique, the qualifier (record-name) must appear AFTER the subscript! For example:

```
AMOUNT (5) IN TRANSACTION
AMOUNT (CODE) OF TRANSACTION
```

It must be remembered that when the level-2 data-name is used, a complete entry will be referenced. When the data-name of one of the sub-items (level-3) is used, only that part of the entry will be referenced.

Subscripting by Decimal Number

Decimal numbers used for subscripting may range from 1 to 999. The number one (1) denotes the first entry in the table, number two (2) specifies the second entry, and so forth. Thus, if the user writes CODE-TABLE (63), he is referring directly to the sixty-third entry in the CODE-TABLE.

Examples:

```
MOVE AGE (11) IN TRANSACTION TO YEARS OF MASTER.
  (Move the eleventh AGE entry in the TRANSACTION record)

ADD AMOUNT (4) AND TOTAL-TO-DATE.
  (Add the fourth AMOUNT entry to TOTAL-TO-DATE)

MULTIPLY PAY BY TAX-RATE (36) GIVING DEDUCTION.
  (Multiply PAY by the thirty-sixth TAX-RATE entry)
```

When subscripting by decimal number, the entry indicated in the statement may not be changed by the programmer. Thus, if the statement is executed a second time, the same table entry will be referenced.

Subscripting by a Data Field

When this method is used, the CONTENTS of the data field will determine which entry is to be referenced. This occurs at object program running time.

Data field subscripting provides the following advantages:

1. In many cases, table searching may be eliminated. For example, if the data field appears as part of a record, the value in the field can vary according to the record being processed. That is, the contents of the data field in one TRANSACTION record may select the sixth table entry, while the same field in another TRANSACTION record may select the twenty-second entry.

2. The programmer may vary the contents of the data-field according to program requirements. He may move values into the field, add to or subtract from it, etc.

Example:

```
MOVE BOND-DEDUCTION (TYPE) TO BONDS.
```

In this example, the decimal number appearing in the field, TYPE, will dictate which table entry is to be moved. If the value in this field is 14, the fourteenth entry in the BOND-DEDUCTION table will be moved; if a three appears, the third entry will be moved, and so forth.

The following rules must be followed when using a data field for subscripting:

1) Field must be UNIQUE. Also, this field cannot itself be subscripted by another data field.

2) Field must be numeric.

3) Field must be fixed in size and cannot exceed three integral digits. Field may include an ISS and/or sign.

4) The value in the field must be an integral value from 1 to 999.

If the value in the data field is zero, or if the value is greater than the number of table entries, an error will occur at program running time. Thus, it is the programmer's responsibility to insure the validity of the number appearing in that field.

5) Constants and record-names (level-1) cannot be used for subscripting purposes.

NOTES

The use of data-name subscripting can eliminate table searching, since the "value" in the data field automatically selects the desired table entry.

Any field, provided it meets the requirements for a subscript field, may be used for subscripting purposes. In addition, the same data field may be used to subscript more than one table.

If desired, the programmer may modify the value within the data field to suit his programming requirements. For example, the contents of the 15 entries in the tax table may be accumulated into the DEDUCTION field by using the data-field, RATE, for subscripting:

```
120. MOVE "1" TO RATE.
130. ADD TAX (RATE) AND DEDUCTION.
140. ADD "1" AND RATE.
150. PERFORM 130 THRU 140 EXACTLY 14 TIMES.
```

EXAMPLE

Assume that an automotive manufacturing company has fourteen district offices, and that the programmer has set up a SALES-STATISTICS file in the following manner:

```
1 GROSS-SALES-BY-DISTRICT; SIZE 56Ø; CLASS NUMERIC.
  2 AUTOMOBILES; SIZE 1Ø; SIGNED; OCCURS 14 TIMES.
  2 TRUCKS; SIZE 1Ø; SIGNED; OCCURS 14 TIMES.
  2 FARM-EQUIP; SIZE 1Ø; SIGNED; OCCURS 14 TIMES.
  2 TOTAL; SIZE 1Ø; SIGNED; OCCURS 14 TIMES.
```

The TRANSACTION-FILE is comprised of individual records which contain appropriate sales figures in the SALES-AUTO, SALES-TRUCK and SALES-FARM fields. In addition, each transaction message contains a two-digit field, DISTRICT, in which a special code (01 to 14) identifies what district is to receive credit for that particular transaction.

The programmer may process these transactions in the following manner:

```
2Ø. READ TRANSACTION.
3Ø. ADD SALES-AUTO AND AUTOMOBILES (DISTRICT).
4Ø. ADD SALES-TRUCK AND TRUCKS (DISTRICT).
5Ø. ADD SALES-FARM AND FARM-EQUIP (DISTRICT).
6Ø. ADD SALES-AUTO, SALES-TRUCKS, SALES-FARM  AND  TOTAL (DISTRICT).
```

Because the programmer has organized his GROSS-SALES-BY-DISTRICT record in table fashion, and has arranged a coding scheme to correspond to table entries (i.e., 01 to 14), a considerable amount of coding and programming effort has been eliminated.

If this approach had not been taken, the program might have appeared as follows:

```
 2Ø. READ TRANSACTION.
 3Ø. IF DISTRICT IS NOT EQUAL TO "Ø1" GO TO 8Ø.
 4Ø. ADD SALES-AUTO AND AUTOMOBILES-1.
 5Ø. ADD SALES-TRUCK AND TRUCKS-1.
 6Ø. ADD SALES-FARM AND FARM-EQUIP-1.
 7Ø. ADD SALES-AUTO, SALES-TRUCK, SALES-FARM AND TOTAL-1; THEN GO TO 44Ø.
 8Ø. IF DISTRICT IS NOT EQUAL TO "Ø2" GO TO 13Ø.
 90. ADD SALES-AUTO AND AUTOMOBILES-2.
1ØØ. ADD SALES-TRUCK AND TRUCK-2.

                    ↓
                   ETC.
```

Note: Four statements required to process each record when "subscripting" was employed; approximately 70 statements required without subscripting.

COPY

Level-number record-name-1; COPY record-name-2.

The COPY option permits the user to automatically duplicate record descriptions that are identical in organization. For example, assume that the organization of the UPDATED-MASTER record is identical to the MASTER record. Instead of defining both records, the user may define the UPDATED-MASTER record as follows:

```
1 UPDATED-MASTER;  COPY MASTER.
```

Because of this entry the Narrator will:

1. duplicate the MASTER record description.
2. change the level-1 name in the duplicated description to UPDATED-MASTER.
3. insert this description for the UPDATED-MASTER record description.

Example:

BEFORE

```
FD MASTER-FILE ......
   1 MASTER .....
     2 EMPLOYEE-NO .....
     2 DEPT-NO .....
     2 NAME .....
FD NEW-MASTER-FILE .....
   1 NEW-MASTER; COPY MASTER.
```

AFTER

```
FD MASTER-FILE .....
   1 MASTER .....
     2 EMPLOYEE-NO .....
     2 DEPT-NO .....
     2 NAME .....

FD NEW-MASTER-FILE .....
   1 NEW-MASTER .....
     2 EMPLOYEE-NO .....
     2 DEPT-NO .....
     2 NAME .....
```

The COPY option can only be used to duplicate complete records -- not files. In addition, the use of this option creates duplicate data-names and qualification must be used in the PROCEDURE DIVISION -- see Qualification, page 30.

No other clauses can appear in this entry.

GENERAL COMMENTS REGARDING RECORD DESCRIPTION ENTRIES

A. LEVEL-NUMBER, DATA-NAME, SIZE and CLASS must appear in this sequence when present in a record description entry. All other clauses within the entry need not be in any special order.

B. Only LEVEL-NUMBER, DATA-NAME and VALUE can appear in a condition-name entry.

C. VARIABLE RECORDS

1) An item is considered VARIABLE when:

   a) its size can vary, or

   b) it is not always present within the record.

2) ALL VARIABLE ITEMS MUST INCLUDE AN ISS SYMBOL.

3) Once an item has been defined as variable, ALL ITEMS FOLLOWING IT WILL ALSO BE CONSIDERED VARIABLE, REGARDLESS OF SIZE.

4) Level-3 entries associated with a variable item cannot be addressed by name in the variable file area. The variable item, however, may be moved to a fixed field which is broken down by sub-items. The programmer may then use sub-item addressing in the fixed area.

D. RECORDS IN MESSAGE FORMAT

When a record appears in "message" format (see page 20), all level-2 items within that record will be assumed to contain an ISS symbol, unless otherwise specified in the PICTURE clause.

E. RECORDS IN BLOCK FORMAT

When a record appears in "block" format (see page 20), all level-2 items within that record will be assumed not to contain ISS symbols, unless otherwise specified in the PICTURE clause.

SAMPLE FILE SECTION

```
DATA DIVISION.

FILE SECTION.

FD MASTER-FILE; LABEL RECORDS ARE STANDARD; VALUE OF ID IS "MØØ5++++";
     DATA RECORDS ARE MASTER.

   1 MASTER; SIZE 6Ø TO 92 CHARACTERS; CLASS ALPHANUMERIC.
     2 ACCOUNT-NUMBER; SIZE 8; PICTURE J.
     2 ACCOUNT-TYPE; SIZE 2; PICTURE J.
     2 DATE; SIZE 7; CLASS NUMERIC.
       3 MONTH; SIZE 2.
       3 DAY; SIZE 2.
       3 YEAR; SIZE 2.
     2 CODE; SIZE 2.
       88 ACTIVE; VALUE IS "1".
       88 INACTIVE; VALUE IS "2".
     2 BALANCE; SIZE 8; SIGNED.
     2 NAME; SIZE 13 TO 27; CLASS AN.
     2 ADDRESS; SIZE 12 TO 22.
     2 CITY-STATE; SIZE 8 TO 16.

FD TRANSACTION-FILE; RECORDING MODE IS BLOCK; LABEL RECORDS ARE OMITTED;
     DATA RECORDS ARE TRANSACTION.

   1 TRANSACTION; SIZE 17; CLASS ALPHANUMERIC.
     2 ACCOUNT-NUMBER; SIZE 8.
     2 TYPE; SIZE 1; CLASS ALPHABETIC.
       88 DEPOSIT; VALUE IS "D".
       88 WITHDRAWAL; VALUE IS "W".
     2 AMOUNT; SIZE 8; CLASS NUMERIC; PICTURE K.

FD NEW-MASTER-FILE; LABEL RECORDS ARE STANDARD; VALUE OF ID IS "MØØ5++++";
     ACTIVE-TIME IS "3"; DATA RECORDS ARE NEW-MASTER.

   1 NEW-MASTER; COPY MASTER.

          EF
```

(Note: If WORKING-STORAGE or CONSTANT SECTION follows, the EF symbol is omitted.)

WORKING-STORAGE SECTION

The Working-Storage Section is optional. When present, it begins with the header "WORKING-STORAGE SECTION." followed by a series of entries describing the working storage areas required for the program. During program running, these areas will be available to all segments of the program.

A working storage entry is written similar to a data record entry, with the following differences:

Non-Contiguous Working Storage Areas

Working storage fields that bear no relationship to one another are called "non-contiguous" working storage. These fields are defined as separate record description entries and are assigned the special level number 77. In the WORKING-STORAGE SECTION, all level 77 entries must be defined first.

Initial Value of Working-Storage Areas

At the start of each program, all working storage areas will be automatically cleared to spaces.

In addition, an Item Separator symbol will be placed in the leftmost position of all level-77 and level-2 fields, except where "PICTURE IS J" has been specified. (ISS's will not be inserted in level-2 fields of records that contain a REDEFINES clause.)

Data-names

Data names assigned to level 77 and level-1 entries must be unique. Level 2 and 3 entries need only be unique within individual records.

Condition names may be associated with the non-contiguous as well as contiguous working storage fields.

Level-Numbers

Level 77 must be assigned to all non-contiguous working-storage areas. The assignment of level numbers to contiguous areas follows the same rules as outlined under the Record Description entry, page 27.

SIZE

All working storage areas must be FIXED in size.

CLASS

All entries defining non-contiguous working storage (level-77) must contain a CLASS clause. The assignment of CLASS for contiguous areas follows the same rules as outlined under the Record Description entry, page 33.

PICTURE IS

It will be assumed that all level-77 and level-2 entries in WORKING-STORAGE contain an ISS. If an ISS is not present in the field, "PICTURE IS J" must be specified.

COPY

The COPY option cannot be used in defining working storage records.

NOTE

All other options appearing in the Record Description section may also be used with working storage areas. See general format, page 26.

REDEFINES

The REDEFINES option is unique to the WORKING-STORAGE SECTION. This feature permits the user to specify that different working-storage records are to share the same memory area. When this option is used, it must immediately follow the record-name being described.

Format:

1 record-name-1 REDEFINES record-name-2 --- etc ---.

This option has the following advantages:

A. It can be used to reduce the amount of working-storage required for the object program. For example, assume that the programmer requires two working-storage records -- WORK-A and WORK-B -- and that he defines them as follows:

```
1 WORK-A; SIZE 8Ø; CLASS AN.
  2 NAME; SIZE 3Ø.
  2 ADDRESS; SIZE 25.
  2 CITY; SIZE 15.
  2 STATE; SIZE 1Ø.

1 WORK-B REDEFINES WORK-A; SIZE 5Ø; CLASS NUMERIC.
  2 TOTAL-A; SIZE 1Ø; SIGNED.
  2 TOTAL-B; SIZE 1Ø; SIGNED.
  2 TOTAL-C; SIZE 1Ø; SIGNED.
  2 TOTAL-D; SIZE 1Ø; SIGNED.
  2 TOTAL-E; SIZE 1Ø; SIGNED.
```

In working storage, the Narrator will assign a common area of 80 characters for these records, instead of reserving an area of 80 characters and an area of 50 characters:

COMMON AREA FOR WORK-A and WORK-B

80 characters

Note: The first 50 characters will be used for the WORK-B record; the entire area will be used for the WORK-A record. Since both records occupy the same memory area, the programmer must exercise caution when processing them. He cannot, for instance, attempt to create both records in this area at the same time. He should first create one record and process it, before attempting to create the next record.

B. The REDEFINES option can be used to specify different formats for the same physical record. For example, assume that WORK-A is defined as follows:

```
1 WORK-A; SIZE 54; CLASS NUMERIC.
  2 COUNTER-1; SIZE 9; SIGNED.
  2 COUNTER-2; SIZE 9; SIGNED.
  2 COUNTER-3; SIZE 9; SIGNED.
  2 COUNTER-4; SIZE 9; SIGNED.
  2 COUNTER-5; SIZE 9; SIGNED.
  2 COUNTER-6; SIZE 9; SIGNED.
```

For programming purposes, the user wishes to address these counters as individual areas. However, at other times he wants the record in table form for subscripting. To accomplish this, he could redefine the record as follows:

```
1 WORK-A-TABLE REDEFINES WORK-A; SIZE 54; CLASS NUMERIC.
  2 TOTAL; SIZE 9; SIGNED; OCCURS 6 TIMES.
```

Memory Layout for WORK-A Record

COUNTER-1	COUNTER-2	COUNTER-3	COUNTER-4	COUNTER-5	COUNTER-6

Memory Layout for WORK-A-TABLE Record

Entry-1	Entry-2	Entry-3	Entry-4	Entry-5	Entry-6

If the programmer desires to move the fourth field in this record area, he may use either of the following methods:

```
MOVE COUNTER-4 TO AMOUNT.

          or

MOVE TOTAL (4) TO AMOUNT.
```

Rules Governing Use of REDEFINES Clause

1. The REDEFINES clause immediately follows the data-name of the entry, with the balance of the entry following the normal rules for describing working-storage. This clause can only be used with level-1 working-storage records.

2. The sizes and formats of each record need not agree.

3. Records that contain a REDEFINES clause will not have Item Separator symbols inserted in level-2 fields at the start of the program.

NUMBER OF WORKING-STORAGE RECORDS

The user may define up to twenty-four (24) records (level-1) in the WORKING-STORAGE SECTION. An error will occur at compilation time if this number is exceeded.

SAMPLE WORKING-STORAGE SECTION

```
WORKING-STORAGE SECTION.
   77 RECORD-COUNT; SIZE 1Ø; CLASS NUMERIC; SIGNED.
   77 NET-BALANCE; SIZE 9; CLASS NUMERIC; SIGNED; PICTURE IS K$999.99S.
   77 ACCOUNT-NO; SIZE 6; CLASS AN.
    1 WORK-TRANSACTION; SIZE 23; CLASS NUMERIC.
      2 CUSTOMER-NO; SIZE 8.
      2 AMOUNT; SIZE 8; SIGNED; POINT LEFT 2.
      2 DATE; SIZE 7.
        3 MONTH; SIZE 2.
        3 DAY; SIZE 2.
        3 YEAR; SIZE 2.
    1 WORK-ERROR; SIZE 7Ø; CLASS AN.
      2 NAME; SIZE 3Ø.
      2 ADDRESS; SIZE 22.
      2 CITY; SIZE 18.
    1 WORK-REORDER; SIZE 34; CLASS NUMERIC.
      2 STOCK-NO; SIZE 6; PICTURE IS J.
      2 ON-HAND; SIZE 8.
      2 REORDER-LEVEL; SIZE 8.
      2 UNIT-PRICE; SIZE 4; ZERO SUPPRESS.
      2 AMOUNT; SIZE 8; SIGNED; ZERO SUPPRESS.
    1 WORK-ERROR-A REDEFINES WORK-ERROR; SIZE 7Ø; CLASS AN.
      2 DATA; SIZE 7Ø; PICTURE IS J.

                 EF
```

(Note: If CONSTANT SECTION follows, EF symbol is omitted.)

CONSTANT SECTION

The Constant Section is optional. When present it begins with the following header: CONSTANT SECTION. This is followed by a series of entries that define the constants associated with the program. During object program running time, these constants will always appear in memory and will be available to all segments.

Constant entries are written similar to data record entries, however, they must conform to the following format:

```
Level-number  data-name

; SIZE IS integer-1 CHARACTERS

              ⎧ AN           ⎫
; CLASS IS    ⎨ ALPHABETIC   ⎬  [; SIGNED]
              ⎪ NUMERIC      ⎪
              ⎩ ALPHANUMERIC ⎭

[; POINT LOCATION IS LEFT integer-2 PLACES]

; VALUE IS literal  .
```

Level-number

The level-number for all constants must be 77.

Data-name

All data-names in the constant section must be unique and an error will occur if identical names appear elsewhere in the Data Division.

SIZE

The SIZE clause must be present and cannot be specified as variable. The size of a constant may not exceed 120 characters.

CLASS

ALL entries in the CONSTANT SECTION must have a class designation; an error will occur if this rule is violated. Classes are determined as follows:

NUMERIC - constant contains the digits Ø-9 (with or without sign)

Example:

```
77 AGE; SIZE 2; CLASS NUMERIC; VALUE IS "32".
77 DATE; SIZE 7; CLASS NUMERIC; VALUE "*Ø13161".
77 BOND-DEDUCTION; SIZE 6; CLASS NUMERIC; SIGNED; VALUE IS "*1875+".
```

(Note that when a sign appears within the constant, the SIGNED clause must also be used.)

ALPHABETIC - constant does not contain a numeric character.

Examples:

```
77 HEADING; SIZE 8; CLASS ALPHABETIC; VALUE IS "*MONTHLY".
77 ERROR; SIZE 4Ø; CLASS ALPHABETIC; VALUE IS
     "TOTAL++++%++++VALUE(NET)++++VALUE(GROSS)".
```

ALPHANUMERIC - constant contains alphabetic and numeric characters, or the constant is a numeric field that includes editing symbols or spaces.

Examples:

```
77 BOND-AMOUNT; SIZE 7; CLASS AN; VALUE IS "*$18.75".
77 DATE; SIZE 8; CLASS AN; VALUE IS "Ø3-14-61".
77 ADDRESS; SIZE 2Ø; CLASS AN; VALUE IS "CLEVELAND,+(14)+OHIO".
77 CREDIT-LIMIT; SIZE 7; CLASS AN; VALUE IS "+++5ØØØ".
```

SIGNED and POINT LOCATION

These clauses are optional and, if used, may only appear for NUMERIC fields. Note that when a sign appears in the field, it must appear as the last character.

VALUE

This clause appears in every constant entry. Note that the size of a constant may not exceed 120 characters, and since this value is surrounded by quotation marks, a quote symbol is not permitted as one of the 120 characters.

Correct	Incorrect
VALUE IS "MONDAY".	VALUE IS ""MONDAY"".

Literals Used In The VALUE Clause

Literals used in the VALUE clause may include all characters listed under LITERALS, page 3. The slant, asterisk and ampersand symbols, however, have the following special connotations in the CONSTANT SECTION:

*	asterisk	-- An asterisk will always be converted to an ISS symbol.
/	slant	-- If a slant appears as the last character in the literal, it will be converted to an EM symbol. In all other positions, slant symbols will produce slant symbols in the constant. (Note that a slant symbol cannot appear alone, or as the first character of a constant.)
&	ampersand	- An ampersand symbol will be converted to a 501 Page Change symbol for off-line printing purposes. The ampersand symbol must appear as a one-character literal ("&").

IMPORTANT NOTE

The maximum area available for constants in NARRATOR programs is 960 characters. The programmer can determine his constant requirements by adding the sizes of each entry in the CONSTANT SECTION.

SAMPLE CONSTANT SECTION

```
CONSTANT SECTION.

      77 CITY-TAX; SIZE 3; CLASS NUMERIC; POINT LEFT 3; VALUE "ØØ5".
      77 STATE-TAX; SIZE 2; CLASS NUMERIC; POINT LEFT 2; VALUE "Ø2".
      77 HEADER-A; SIZE 24; CLASS ALPHABETIC; VALUE "MONTHLY+SALES+STATISTICS".
      77 HEADER-B; SIZE 23; CLASS ALPHABETIC; VALUE "WEEKLY+SALES+STATISTICS".
      77 ISS; SIZE 1; CLASS AN; VALUE IS "*".
      77 MESSAGE-FILL; SIZE 6; CLASS AN; VALUE IS "*****/".
      77 PRINT-LINE; SIZE 12Ø; CLASS AN; VALUE IS
           "*++++++++++*+++++*++++++++++++++++++*+++++*++++++++++++*++++*
            ++++++++*+++++++++++++++++++++++++*++++++++++++*+++++++++*++".

                        EF
```

THE PROCEDURE DIVISION

The PROCEDURE DIVISION specifies the steps that the user wishes the computer to follow in order to produce the desired results. These steps are expressed in meaningful English statements which include the concept of verbs to denote actions, sentences to desrribe procedures and IF statements to provide alternative paths of action.

This aspect of the Narrator system is often referred to as the "program," but by itself is not sufficient to describe the entire problem. This is true because repeated references must be made -- either implicitly or explicitly -- to information appearing in other divisions.

HEADER

The Procedure Division begins with the following header: PROCEDURE DIVISION. It is terminated by an EF symbol.

Note: If the input information for the Narrator ends with the procedural statements, i.e., there is no own-coding, an ED symbol follows the EF symbol.

SECTIONS

The largest unit in the PROCEDURE DIVISION is a section, which identifies points at which the object program is to be segmented. Each section is identified in the following manner:

line-number SECTION.

The Narrator treats each section as being a segment of the object program. The absence of SECTION headers indicates that the PROCEDURE DIVISION is one section and that the object program is not to be segmented. Segmentation is discussed on page 118.

SENTENCES

The Procedural statements are expressed in a manner similar (but not identical) to normal English prose. There are two types of statements:

A. SIMPLE IMPERATIVE SENTENCES

Examples: READ MASTER-FILE.
SUBTRACT DEDUCTIONS FROM PAY.

B. SIMPLE CONDITIONAL SENTENCES

Examples: IF BALANCE IS LESS THAN ZERO GO TO 46.
IF DATE OF MASTER EQUALS REVIEW-DATE GO TO 15.

LINE NUMBERS

Each Procedural statement is identified by a line number which may be any decimal number ranging from 1 to 99999, followed by a period. For sequencing purposes the Narrator sorts the procedural statements into ascending line number order. It is not a requirement that the initial program be written in line number sequence, although it must be remembered that the program will operate in sequential order.

Line numbers do not have to be in consecutive order and it is permissable to "skip" numbers. This facilitates applying additions and corrections to the program. For example, assume that the original program was written in the following order: 10, 20, 30, 40, 50, 60, etc. If the user now desires to add three statements following line number 20, he could insert them as line numbers 21, 22 and 23.

Methods of making additions and corrections are covered in a separate publication.

USE OF LITERALS WITHIN THE PROCEDURE DIVISION

The format for most Narrator verbs permits the user to write literals within statements. For example:

```
MOVE "5" TO CODE.
ADD "6Ø" AND BALANCE.
IF AREA-CODE IS EQUAL TO "ABC" GO TO 317.
```

When used within statements, literals must be enclosed by quotation marks and may not exceed 120 characters. (The quote marks are not included in computing the size.)

User-Specified literals

The user may create literals from the character set listed on page 3. Note that the slant and asterisk are not acceptable characters for literals used in the PROCEDURE DIVISION.

The Narrator will "expand" user-specified literals if required in the destination area. That is, ISS symbols, positive signs, zeros or spaces will be added to the literal to conform with the description of the receiving field. For example, if the literal "5" is moved to a three-character field that includes an ISS and sign location, the literal will be expanded to "•5+".

Narrator-generated Literals

The Narrator will create literals to accommodate the following cases:

a) Use of Figurative constants ZERO or ZEROS.

b) Testing condition-names.

c) Moving data into larger-sized fields.

d) Alignment of decimal locations.

e) Editing symbols and editing functions (BLANK WHEN ZERO, etc.)

Number of Literals

Each SECTION of the program may contain up to 74 literals that are 17 characters or less, and 40 literals that range in size from 18 to 120 characters. These include both user-specified literals as well as those literals generated by the Narrator. (Note that identical literals within each section will not be duplicated in the object program and that duplicated literals are not included in the above count.)

Class of Literals

At compilation time, all literals within procedural statements are examined and assigned a "CLASS," based on the normal rules for determining CLASS (see page 33). Thus, the literal "12-15-61" would be assigned an alphanumeric class; and an error would be indicated if the user attempts to add this literal to a numeric field.

THE ALL OPTION

The Narrator user may precede a literal with the word ALL, in which case the Narrator will generate that literal as a string of homogenous information.

Example A: MOVE ALL "4" TO FIELD.

```
   Before                 After

- - - - - - - - - -    4 4 4 4 4 4 4 4 4 4
```

(The literal generated for this statement would be: "4444444444".)

Example: MOVE ALL "FOUR" TO FIELD.

```
   Before                 After

- - - - - - - - - -    FOURFOURFO
```

(Note that the literal is truncated when the field is filled. The generated literal for this example would be: "FOURFOURFO".)

FIGURATIVE CONSTANTS

The Narrator will also accept the following "figurative" constants when the user desires to specify the actual value of a literal as being "zero" or "space": ZERO, ZEROS, ZEROES, SPACE and SPACES.

```
Examples:  IF BALANCE IS EQUAL TO ZERO GO TO 318.
           MOVE ZEROS TO AMOUNT.
           MOVE SPACES TO PRINT-AREA.
```

Note that "figurative" constants are not enclosed with quotation marks and may only be used within the PROCEDURE DIVISION. Also, the use of ZERO(S) produces a "generated" literal equal to the number of data characters in the corresponding field.

QUALIFICATION

In the discussion of data-names in the Record Description, page 30, it was stated that duplicate data-names may be used as long as they do not appear within the same record. When the user desires to refer to a duplicate data-name in the procedural statements, he must qualify that name by the record in which it appears. This is accomplished by following the data-name being addressed with the word OF or IN, followed by the record-name in which it appears. For example:

```
ADD AMOUNT OF MASTER AND BALANCE.

ADD AMOUNT IN TRANSACTION AND BALANCE IN MASTER.

IF TRANS-CODE IS LESS THAN CODE IN MASTER GO TO 47.
```

Note that the use of the COPY option in the DATA DIVISION creates duplicate data-names.

VERBS

Verbs available to the Narrator user are listed under the following categories:

Arithmetic	Input-Output	Procedure Branching
ADD	READ	GO
SUBTRACT	WRITE	ALTER
MULTIPLY	DISPLAY	PERFORM
DIVIDE	OPEN	FIND
	CLOSE	*IF
	ACCEPT	

Compiler Directing	Data Movement	Ending
ENTER	MOVE	STOP
EXIT	EXAMINE	
NOTE		
USE		

*Although the word IF is not a verb in the strictest sense, it possesses one of the most important characteristics of one -- namely, the generation of coding in the object program. The IF statement is a vital feature in the PROCEDURE DIVISION, and is fully discussed in this section under IF Statements.

RULES GOVERNING ARITHMETIC OPERATIONS

The following rules apply to all arithmetic operations -- ADD, SUBTRACT, DIVIDE and MULTIPLY.

1. OPERANDS

   All operands used in arithmetic operations MUST BE NUMERIC. Operands CANNOT contain editing symbols ($.,space), nor should they be specified as "edited," i.e., ZERO SUPPRESS, CHECK PROTECT, FLOAT DOLLAR SIGN or BLANK WHEN ZERO.

   The maximum size of any operand is 18 digits, which includes both integer and decimal locations, but not including ISS symbol or sign.

   An arithmetic operand cannot be a record-name or a condition-name.

STORAGE OF RESULT

a) The result field must have standard justification.

b) Constants, literals, condition-names and records cannot be result fields.

c) If the GIVING clause is not specified, the result will be stored in the last operand. For example:

   ADD A, B, C, D, E, F.

   (Accumulated sum will be stored in "F" field.)

d) If the GIVING option is used, the result will be stored in the "data-name" specified. For example:

   MULTIPLY A BY B GIVING C.

   (Product will be stored in "C" field.)

Special rules governing GIVING field:

1. The result stored in the GIVING field replaces whatever appeared in that field previously. It is not necessary, therefore, to clear the GIVING field to zero. (Note that the result is not accumulated to the previous value.)

2. The GIVING field is the only field in an arithmetic operation that may be "edited." When this field is to be edited, it cannot contain more than eighteen numeric characters, exclusive of editing characters and symbols which are to be inserted. For rules governing editing, see pages 40 through 45. Note that editing symbols cannot be expressed in the PICTURE clause by parenthentical notation.

5. TRUNCATION OF DECIMAL LOCATIONS

When the result field contains fewer decimal locations than the computed result, automatic truncation of decimal locations occurs. For example, assume that in the following operation the result field had been defined as having two decimal positions:

```
Operand A    13.6666
Operand B     6.22           Result Field
             19.8866 ——————> 19.88
```

(Note that decimal points are assumed.)

6. ROUNDING

When truncation of decimal locations occurs and the ROUNDED option has also been specified, a "one" is added to the least significant digit of the result if the most significant digit of the decimal excess is "five" or greater. If this digit is less than "five," no rounding occurs.

Thus, in the preceding example, if ROUNDED had been specified, the result field would have been $19_{\wedge}89$.

If decimal locations are not truncated, the ROUNDED clause is redundant and will have no effect on the result.

7. EDITING

As stated previously, operands in arithmetic operations cannot be "edited." However, the programmer may obtain an "edited" result by using the GIVING option (provided the GIVING field is not also one of the operands).

To illustrate this point, let us assume that the programmer wrote the following statement:

ADD A, B, C, GIVING D.

The computed result of this operation amounted to 477.33 (two assumed decimal locations). This result would be placed in field "D" as follows:

A) FIELD-D; SIZE 7; POINT LEFT 2; PICTURE K999.99.

•	4	7	7	.	3	3

B) FIELD-D; SIZE 9; POINT LEFT 2; PICTURE J$9,999B99.

$	Ø	,	4	7	7	-	3	3

C) FIELD-D; SIZE 9; POINT LEFT 2; PICTURE J$9,999B99; ZERO SUPPRESS.

$	-	-	4	7	7	-	3	3

D) FIELD-D; SIZE 8; ZERO SUPPRESS.

•	-	-	-	-	4	7	7

3. USE OF LITERALS

Literals cannot exceed eighteen numeric characters. Figurative constants or the "ALL any-literal" option cannot be used.

4. TRUNCATION OF INTEGERS

a) An error warning will be indicated at compilation time when the number of integral locations in the result field cannot accommodate the integral digits in the computed result. For example, if a field with four integers is added to a field with two integers, the result field should contain at least four integral locations.

When the result field does not contain sufficient integral locations, the excess integral digits will automatically be truncated. For example, assume that the result has two integral places:

$$\begin{array}{lrl} \text{Operand A} & 61744.22 & \\ \text{Operand B} & \underline{5781.11} & \qquad \text{Result Field} \\ & 67525.33 & \longrightarrow 25.33 \end{array}$$

(Note that decimal points are assumed)

b) A potential "overflow" condition can exist even though the size of the result field indicates that it can accommodate integers of the computed result. For example, a two-integer result field can accommodate the following addition:

$$\begin{array}{lr} \text{Operand A} & 44 \\ \text{Operand B} & \underline{23} \\ & 67 \end{array}$$

However, the data may be such that at object program running time a one-digit overflow could occur. For example:

$$\begin{array}{lr} \text{Operand A} & 44 \\ \text{Operand B} & \underline{72} \\ & \boxed{1}16 \\ \text{overflow} \nearrow & \end{array}$$

Therefore, when overflow in the result field is dependent on the data at object program running time, the ON SIZE ERROR option should be used. This option will cause truncation of the overflow digit and a transfer to the designated line-number when such a condition occurs. If overflow arises, and this option has not been specified, the result will be unpredictable.

Note: Since the ON SIZE ERROR option generates additional coding in the object program, its use is recommended only when the integral digits to be stored in the result field may exceed the size of that field.

ACCEPT VERB

Function: To enter into the computer low volume information from magnetic tape or the Paper Tape Reader.

$$\underline{\text{ACCEPT}}\ \text{record-name}\ \underline{\text{FROM}} \begin{Bmatrix} \underline{\text{PAPER-TAPE-READER}} \\ \underline{\text{TRUNK}}\text{-number} \end{Bmatrix} .$$

When an ACCEPT statement is to be executed in the object program, a message to this effect will be displayed on the Monitor Printer, and the program will stop to allow for mounting this data on the appropriate equipment. Depressing the start button will then cause this information to be read into the record-name area, and the program will be resumed.

NOTES:

1. "Record-name" may be either a level-1 file area or a level-1 working-storage area. "Number" is written as two characters-Ø6, 5Ø, etc.

2. The data to be entered must appear in message format.

3. If a file area is specified, the record must be defined as being in message format.

4. If a working-storage area is specified, the user must include the SM and EM symbols in the size of the working-storage area, and must also define these symbols as items within the record.

   For example, assume that today's date is to be accepted from the Paper Tape Reader into working-storage:

```
1 DATE; SIZE 8; CLASS NUMERIC.
  2 SM; SIZE 1; PICTURE J.
  2 TODAYS-DATE; SIZE 6; PICTURE J.
  2 EM; SIZE 1; PICTURE J.
```

   (Today's date may now be directly referenced by name.)

EXAMPLES

a) ACCEPT DATE FROM PAPER-TAPE-READER.

b) ACCEPT LIABILITY-TABLE FROM TRUNK-Ø3.

ADD VERB

Function: To add two or more quantities and store the sum in either the last named field or in the specified one.

$$\text{ADD} \left\{\begin{matrix}\text{literal-1}\\ \text{data-name-1}\end{matrix}\right\} \left\{\begin{matrix},\\ \underline{\text{AND}}\end{matrix}\right\} \left\{\begin{matrix}\text{literal-2}\\ \text{data-name-2}\end{matrix}\right\} \left[\left\{\begin{matrix},\\ \underline{\text{AND}}\end{matrix}\right\} \left\{\begin{matrix}\text{literal-n}\\ \text{data-name-n}\end{matrix}\right\} \ldots\right]$$

$$\left[\underline{\text{GIVING}}\ \text{data-name-m}\right] \quad \left[\underline{\text{ROUNDED}}\right] \quad \left[;\ \text{ON}\ \underline{\text{SIZE}}\ \underline{\text{ERROR}}\ \underline{\text{GO}}\ \underline{\text{TO}}\ \text{line-number}\right] .$$

NOTES:

1. See RULES GOVERNING ARITHMETIC OPERATIONS, Page 66. Note that when the integer locations of the result field equals the integer locations of the largest operand, a possible "overflow" condition may exist.

2. In addition to the above rules, the following apply:

   a) A maximum of 10 operands can be specified in any single ADD statement. This includes the GIVING field, if present.

   b) Literals must contain the same number of decimal locations as the result field. If, for example, the literal "5ØØ" has been used, and the result field has two decimal locations, the literal is considered as being 5‸ØØ. If the result field has no decimal locations, the literal is considered to be 5ØØ‸.

EXAMPLES

```
 4Ø. ADD BALANCE AND NET.
 7Ø. ADD AMT-A, AMT-B, AMT-C, AMT-D.
13Ø. ADD COST, MFG-EXPENSE GIVING NET-COST ROUNDED.
44Ø. ADD SALE, TAX, SHIPPING GIVING AMOUNT; ON SIZE ERROR GO TO 582.
```

ALTER VERB

Function: To change the line-number appearing in a GO TO statement. (The statement being modified must be an independent GO TO statement that contains only one destination address.)

ALTER line-number-1 TO PROCEED TO line-number-2.

"Line-number-1" specifies the GO TO statement to be modified; "line-number-2" indicates the NEW destination address that is to appear in that statement.

The GO TO statement is frequently used as a switch which the programmer can "set" to alternate among various processing paths. The ALTER statement, therefore, provides the user with a convenient method of setting GO TO statements to particular processing paths. For example, assume that the GO statement at line 4Ø7 is being used by the programmer as a "switch."

```
   .
   .
   .
  4Ø. ALTER 4Ø7 TO PROCEED TO 51Ø.
   .
   .
   .
 27Ø. ALTER 4Ø7 TO PROCEED TO 53Ø.
   .
   .
   .
   .
 33Ø. ALTER 4Ø7 TO PROCEED TO 56Ø.
   .
   .
   .
 4Ø7. GO TO 5ØØ.
   .
   .
   .
```

Note that when a GO TO address is modified it will retain that address until changed by another ALTER statement. Also, a GO TO statement in one SECTION cannot be changed by an ALTER statement within another SECTION -- see Segmenting, page 118.

CLOSE VERB

Function: To terminate the processing of input and output reels and files.

```
CLOSE file-name-1   [REEL]  [WITH NO REWIND]

[ , file-name-2     [REEL]  [WITH NO REWIND] ...ETC..... ] .
```

CLOSE file-name

Examples:

CLOSE MASTER, TRANSACTION, ERROR, NEW-MASTER.
CLOSE MASTER.
CLOSE NEW-MASTER.

A. INPUT FILES

When an Input file is closed, the tape is rewound to BTC, unless the NO REWIND option has been specified. If the programmer closes an Input file which has not been completely processed, the End Label checking procedure will be bypassed.

B. OUTPUT FILES

When an Output file is closed, the End Label procedures are performed, if required. See Tape Labels, page 126. The end of file sentinels (i.e., EF and ED) are then written and the tape is rewound to BTC, unless NO REWIND has been indicated.

(The above procedures do not occur when a PRINT file is closed.)

CLOSE file-name REEL:

Examples:

CLOSE SALES-FILE REEL.
CLOSE REPORTS-FILE REEL.

When a file contains multiple reels, the opening and closing of individual reels is automatically performed by the Input/Output Control. It is only required that the user close the file when processing is to be terminated. If, however, the user desires to close individual reels, the following events will occur:

A. INPUT files:

NO End Label checking will be performed. The tape will be rewound to BTC, unless the NO REWIND option has been specified. If NO REWIND is used, the tape will be positioned after the ED file sentinel.

Note: CLOSE REEL cannot be used for an input file that is being used as the re-run control file. If this is attempted, an error will be indicated at compilation time.

(Cont'd)

CLOSE file-name REEL: (Cont)

B. OUTPUT files:

The End Label is created and written to the output file -- see Tape Labeling. An ED file sentinel is then written to the file and the tape rewound to BTC, unless NO REWIND has been specified. The next reel for this file is then opened following the standard procedure for this file.

NOTE

A "CLOSE file-name" must appear for each file that has been "opened" in the program. A file must be closed before it can be reopened. If an OPTIONAL file is not present during object program running time all CLOSE statements for that file will be ignored.

EXAMPLES

```
CLOSE MASTER-FILE, TRANSACTION-FILE WITH NO REWIND, NEW-MASTER-FILE.
CLOSE ERROR-FILE.
```

DISPLAY VERB

Function: To allow for visual display of low volume information on the monitor printer.

$$\underline{\text{DISPLAY}} \left\{ \begin{array}{l} \text{literal-1} \\ \text{data-name-1} \end{array} \right\} \left[, \left\{ \begin{array}{l} \text{literal-2} \\ \text{data-name-2} \end{array} \right\} \quad \text{etc....} \right] \ \underline{\text{UPON}}\ \underline{\text{MONITOR}} \ .$$

Each DISPLAY statement will position the monitor printer on the next print line, and printing will start at the left-hand margin. As indicated in the format, multiple literals or data-names may be included in a single DISPLAY statement.

It a space is desired to separate literals or data fields, the programmer may use the figurative constant,"SPACE." (No other figurative constants are permitted.)

EXAMPLES

```
DISPLAY "INVALID STOCK NUMBER" UPON MONITOR.
DISPLAY "OVERDRAFT" SPACE, ACCOUNT-NO OF MASTER UPON MONITOR.
DISPLAY COUNTER-1, SPACE, COUNTER-2, SPACE, COUNTER-3 UPON MONITOR.
```

DIVIDE VERB

Function: To divide one number by another and to store the result in the last named field or the specified one.

```
DIVIDE  { literal-1   }   INTO   { literal-2   }
        { data-name-1 }          { data-name-2 }

[GIVING data-name-3]   [ROUNDED]   [; ON SIZE ERROR GO TO line-number].
```

NOTES:

1. See RULES GOVERNING ARITHMETIC OPERATIONS, page 66.

2. In addition to the above rules, the following apply:

   a) Literals used in the DIVIDE statements must be integer values (whole numbers).

   b) If the value of the divisor is "zero," a machine alarm will occur. Therefore, if this condition could exist, the user must test for a "zero" divisor and, if present, avoid the execution of the DIVIDE statement.

EXAMPLES

```
 16. DIVIDE TOTAL-SALES INTO DEPARTMENT-A GIVING PERCENTAGE.
117. DIVIDE "1ØØ" INTO AMOUNT GIVING TAX-BASE ROUNDED.
412. DIVIDE UNITS-ON-HAND INTO INVENTORY GIVING UNIT-COST.
```

ENTER VERB

Function: To permit the user to incorporate coding written in RCA 501 Automatic Assembly format into the Narrator program.

ENTER ASSEMBLER P-number.

The explicit P-address of the FIRST instruction in the block of Assembly to be entered is supplied as the "P-number." The format to be followed in writing 501 Automatic Assembly coding is discussed in the Own Code Section, page 120.

Assume that the programmer wishes to include 501 Assembly coding following statements 40 and 80:

```
 .
 .
4Ø. MOVE BALANCE IN MASTER TO WORK-BALANCE.
41. ENTER ASSEMBLER PAC1Ø.
42. WRITE NEW-MASTER.
 .
 .
8Ø. ADD BALANCE IN MASTER AND NEW-BALANCE.
81. ENTER ASSEMBLER PAC1Ø.
82. WRITE ERROR.
 .
 .
```

The 501 Assembly coding will be inserted into the object program at the place where it is called, i.e., in line. A second ENTER statement, having the same P-number, will produce another copy of this Assembly coding in the program.

Once a block of Assembly coding has been entered, further references of this coding can be made through PERFORM or GO TO statements, that refer to the line-number of the ENTER statement. For example:

```
  .
  .
 4Ø. MOVE BALANCE IN MASTER TO WORK-BALANCE.
 41. ENTER ASSEMBLER PAC1Ø.
 42. WRITE NEW-MASTER.
  .
  .
  .
 8Ø. ADD BALANCE IN MASTER AND NEW-BALANCE.
 81. PERFORM 41.
 82. WRITE ERROR.
  .
  .
  .
14Ø. GO TO 41.
  .
  .
```

If PERFORM is used, control will be transferred to the Assembly coding and returned to the statement immediately following the PERFORM line. If GO TO is used, control will be transferred to the ENTER line and processing will continue "in-line" from that point.

NOTE: Up to 50 ENTER statements may appear in the program.

EXAMPLES

```
ENTER ASSEMBLER PAS1Ø.

ENTER PARØØ.
```

EXAMINE VERB

Function: The EXAMINE verb has two purpoees:

a) REPLACING: to replace a specific character in a data field by another specified character.

b) TALLYING: to count the number of times a specific character appears in a data field. (This option can also be used to replace these occurrences, if desired.)

```
                         ┌ ALL           ┐
            ┌ REPLACING  │ LEADING       │  literal-1 WITH literal-2 .        ┐
EXAMINE data-name        └ [UNTIL] FIRST ┘                                    │
            │            ┌ ALL           ┐                                    │
            └ TALLYING   │ LEADING       │ literal-3 [REPLACING WITH literal-4] . ┘
                         └ UNTIL FIRST   ┘
```

NOTES:

1. The literal must be a single character, or the "figurative" constants ZERO(S) and SPACE(S) may be used. For example:

   EXAMINE NUMBER REPLACING ALL "A" WITH "B".

   EXAMINE NUMBER REPLACING LEADING ZEROES WITH SPACES.

2. The examining process always starts from the left-hand end of the field. (Note that an Item Separator before the field is NOT considered as the LHE.)
3. If UNTIL FIRST is used with the REPLACING option, all characters up to the first occurrence of literal-1 will be replaced by literal-2. For example:

   EXAMPLE FIELD-A REPLACING UNTIL FIRST "5" WITH ZERO.

Before	After
|•|4|4|7|5|7|2|	|•|Ø|Ø|Ø|5|7|2|

   Note that if the literal-1 character did not appear in the field, the entire field would have been changed to the literal-2 character.

4. When the TALLYING option is used, the tally "count" is automatically placed in a special data field called TALLY. TALLY is a 7-character field containing an ISS symbol and positive sign, which the user may directly address by name. The "count" may vary from 00000 to 99999. The name, TALLY, cannot be used for any purpose other than to address this special field.

Example:

20. EXAMINE AMOUNT TALLYING ALL ",".

AMOUNT FIELD	TALLY (BEFORE)	TALLY (AFTER)
$ 3 , 4 1 1 , 2 2 . 1 5	• Ø Ø Ø 3 6 +	• Ø Ø Ø Ø 2 +

Note that each time the tallying function is used, the previous contents of the TALLY field are destroyed. If the contents are to be retained, the user should move the field to another area.

If the ALL option is used, the tally count will be the number of occurrences of "literal-3." When LEADING is used, the count will be the number of occurrences prior to encountering a character other than "literal-3." If UNTIL FIRST is specified, the tally count will be the number of characters encountered before "literal-3."

EXAMPLES

a) EXAMINE AMOUNT REPLACING LEADING ZEROES WITH SPACES.

Before	After
• Ø Ø Ø 8 4 4 -	• - - - 8 4 4 -

b) EXAMINE STOCK-NO REPLACING FIRST "A" WITH "B".

Before	After
4 6 6 - A 7 3 2	4 6 6 - B 7 3 2

c) EXAMINE NAME TALLYING UNTIL FIRST SPACE.

NAME FIELD	TALLY (BEFORE)
J O E - - - - -	• Ø Ø Ø Ø 5 +

TALLY (AFTER)
• Ø Ø Ø Ø 3 +

d) EXAMINE CODE TALLYING ALL "A" REPLACING WITH "B".

CODE FIELD (BEFORE)	TALLY (BEFORE)
A 8 6 6 A C	• Ø Ø Ø 1 3 +

CODE FIELD (AFTER)	TALLY (AFTER)
B 8 6 6 B C	• Ø Ø Ø Ø 2 +

EXIT VERB

Function: To furnish a common exit point for a subroutine.

EXIT.

The EXIT statement permits the user to interrupt the normal sequence of execution within a subroutine and to go directly to the "exit" of that subroutine. This verb produces NO coding in the object program and is only used to provide a method of leaving the subroutine without executing any more statements in the subroutine.

The EXIT statement must appear as the last line of the subroutine.

Example A

```
PERFORM   ┌─ 2Ø. MOVE A TO B.
RANGE     │  3Ø. IF B IS GREATER THAN "5ØØ" GO TO 6Ø.
          │  4Ø. ADD B AND WORK-AMOUNT.
          │  5Ø. MOVE WORK-AMOUNT TO PRINT-AMOUNT.
          └─ 6Ø. EXIT.
```

In this example the program will leave the perform range when statements 20 through 50 are executed, or if field B is greater than "500."

Example B

```
             ┌─ 4ØØ. IF SECURITY-CODE IS EQUAL TO "C" GO TO 44Ø.
USE          │  41Ø. IF SECURITY-CODE IS EQUAL TO "R" GO TO 44Ø.
SUBROUTINE   │  42Ø. IF SECURITY-CODE IS EQUAL TO "S" GO TO 44Ø.
             │  43Ø. STOP "SECURITY CODE INVALID".
             └─ 44Ø. EXIT.
```

If a valid security code appears in the tape label, the programmer leaves the USE Subroutine by transferring to the common exit line.

NOTES

1. Control is transferred to the line-number of the EXIT statement.

2. EXIT lines are initially set to go to the next statement in sequence. However, when the subroutine is executed as a PERFORM range, the exit will be reset to transfer to the statement following the PERFORM line. See PERFORM verb for further clarification of the process of setting and resetting the BY-PASS.

FIND VERB

Function: To determine the relationship that exists between a specified data field and entries that appear in a list or table.

```
FIND data-name WITHIN table-name [(subscript-name)]

 ;IF relation-1 THEN GO TO line-number-1

[ ; { IF relation-2 THEN }                      ]
[   { OTHERWISE          } GO TO line-number-2  ] .
[   { ELSE               }                      ]
```

The FIND verb enables the user to compare the value in a data field against entries that appear in a table or list, according to the following relations:

```
[NOT] EQUAL TO       UNEQUAL TO

[NOT] LESS THAN      EXCEEDS

[NOT] GREATER THAN   EQUALS
```

Example:

```
2Ø.FIND CODE WITHIN CODE-TABLE; IF EQUAL TO GO TO 43Ø.
```

The value in the data field, CODE, will be compared to the value in the first entry in the CODE-TABLE. If these values are not equal, the data field will then be compared to the second entry, and so on. This comparison process continues until:

1) Equality is found. (At this point the program will transfer to line 430.)

2) The entire table has been searched without finding an equal condition. (When this occurs the next statement in sequence will be executed.)

Using "Subscript-name" Option

If the (subscript-name) option is used, the Narrator will store the relative position of the table entry that satisfied the condition in the "subscript-name" field.

Example:

```
2Ø.FIND CODE WITHIN CODE-TABLE (LOCATION); IF EQUAL TO GO TO 43Ø.
```

In this case, each time the data field, CODE, is compared to a table entry, the Narrator will add a "1" to the data field, LOCATION. Thus, if the 18th entry in the table was equal to the CODE field, the data field LOCATION would contain the value "18." (The programmer may now use the LOCATION field for subscripting purposes -- see Subscripting, page 47 .)

"Table-name"

The user substitutes for "table-name" the level-2 data name that contained the OCCURS clause, if he wishes to reference the entire entry. If only part of the entry is to be referenced, the data-name of one of the level-3 sub-items is used. For example:

```
2 ANNUAL-EXPENDITURE; SIZE 1Ø; PICTURE J; OCCURS 3Ø TIMES.
  3 YEAR; SIZE 2.
  3 AMOUNT; SIZE 8.
```

```
41Ø. FIND TRANS-AMT WITHIN ANNUAL-EXPENDITURE, IF EQUAL TO GO TO 42.
```

(In this case, the TRANS-AMT field will be compared to the full 10 characters of each entry in the ANNUAL-EXPENDITURE table.)

```
32Ø. FIND TRANS-YEAR WITHIN YEAR; IF EQUAL TO GO TO 2Ø7.
```

(In this case, the TRANS-YEAR field will be compared to the first two characters of each table entry.)

```
2Ø3. FIND TRANS-AMOUNT WITHIN AMOUNT; IF EQUAL TO GO TO 24.
```

(In this case, the TRANS-AMOUNT field will be compared to the last 8 characters of each table entry.)

By using the "subscript-name" option in the FIND statement, the programmer may take advantage of the table location (which is automatically stored for him) to process other parts of the same entry. For example, assume that the following statements appeared in the program:

```
4Ø. FIND TRANS-YEAR WITHIN YEAR (LOCATION); IF EQUAL TO GO TO 33Ø.

33Ø. MOVE AMOUNT (LOCATION) TO WORK-AMOUNT.
```

After statement 40 has been executed, the subscript field LOCATION will contain the relative number of the entry that was equal to TRANS-YEAR. (If this happened to be the seventh entry, then a "7" would appear in LOCATION.)

The programmer then uses the LOCATION field to specify which AMOUNT entry is to be used in the MOVE operation. In effect, the user is saying "Move the seventh amount entry to WORK-AMOUNT."

NOTES

1. When the relations EQUAL TO or EQUALS are used by themselves, the entire table will be searched until equality is found or the end of the table has been reached. For example, assume that the field CODE contains the value 08, and that CODE-TABLE has fifty entries ranging in value from 01 to 50:

```
6Ø. FIND CODE WITHIN CODE-TABLE; IF EQUAL TO GO TO 42.

7Ø. MOVE .... .
```

In this case, 08 will be compared to each table entry until an "equal" is found, at which time a transfer will be made to statement 42. If 08 is not one of the values in the table, the next statement in sequence will be performed.

2. ALL OTHER RELATIONS (NOT EQUAL TO, UNEQUAL TO, LESS THAN, NOT LESS THAN, etc.) WILL TERMINATE THE TABLE SEARCH AT THE FIRST TABLE ENTRY THAT COMPLIES WITH THE RELATION SPECIFIED. For example, assume that the field PAY contains the value 075, and that the SALARY-RANGE table contains ascending values starting with 050, 060, 070, etc., up to 200:

```
60. FIND PAY WITHIN SALARY-RANGE (RATE); IF
    LESS THAN GO TO 53; OTHERWISE GO TO 174.
```

In this example, "075" will be compared to each entry in the list -- starting with the first -- until this value is less than the list entry. This condition will occur when the fourth entry (080) in the list is compared. At this time the subscript field, RATE, will contain the value "4," and control will be transferred to statement 53.

The FIND statement may be used to search one table in order to gain entrance to another table. For example, assume that a TAX table contains the corresponding tax rate for each entry in the SALARY-RANGE table. In this case, the value appearing in the subscript field, RATE, would now contain the number of the entry in the TAX table that affects that salary range. Thus, statement 53 might appear as follows:

```
53. MULTIPLY PAY BY TAX (RATE) GIVING DEDUCTION.
     (That is, multiply PAY by the 4th TAX entry.)
```

3. "Data-name" and "table-name" must contain the same number of data characters and must be fixed in size.

GO VERB

Function: To interrupt the normal sequence of statements by specifying the next statement to be performed.

OPTION 1

$\underline{\text{GO}}\ \underline{\text{TO}}\ \left[\text{line-number}\right].$

This option permits the user to specify that processing is to be transferred to the designated statement, instead of the next sequential statement. For example, let us assume that the programmer wishes to transfer processing to statement 410 after line 40 has been executed:

```
4Ø. MOVE A TO B.
5Ø. GO TO 41Ø.
6Ø. MOVE A TO C.
```

It may be desired to omit the line number in a GO TO statement when it is being used as a program "switch." In these cases, the ALTER verb must be used to set this "switch" to a specific line number when desired. (If this switch is never set by an ALTER statement, an error will occur in the object program.)

Example:

```
  .
  .
 54. ALTER 41Ø TO PROCEED TO 5ØØ.
  .
  .
  .
41Ø. GO TO.
  .
  .
```

OPTION 2

$\underline{\text{GO}}\ \underline{\text{TO}}\ \text{line-no-1, line-no-2}\left[\text{,line-no-3,...etc...}\right]\ \underline{\text{DEPENDING}}\ \text{ON data-name.}$

This option permits the user to designate a number of destinations in a GO TO statement. The actual line-number to which the program will transfer is determined by the numeric value appearing in the "data-name" field.

Example:

```
4Ø. GO TO 314, 62, 18Ø, 2Ø, 3Ø2, 2Ø, 178, 54Ø DEPENDING ON TRANS-CODE.
```

If the field "TRANS-CODE" contains the value "3" when statement 40 is executed, control will be transferred to line 180. If the value "6" is present, the program will transfer to line 20.

The following rules apply when using the GO TO DEPENDING ON option:

1. Up to 60 line numbers may be specified in the statement.

2. "Data-name" field:

   a) may be any level-2 or level-3 entry, or a level-77 working-storage area. (Constants and condition-names cannot be used for this purpose.)

   b) The size of the data-name field must always be constant (fixed). This field may be defined as being one, two or three characters (not including an ISS, if present). The field cannot contain a sign position.

   c) The value in the data-name field must be an integer value which is right-justified and zero-filled. Also, since the value in this field determines the destination address, this value should range between 1 and the total number of addresses appearing in the GO TO statement. <u>If the value in the data field</u> is zero, or it exceed <u>the number of destination areas, the next statement in sequence will be performed</u>.

<u>SPECIAL OPTION</u>

[; THEN <u>GO</u> <u>TO</u> line-number]

The optional clause "THEN GO TO" may be appended to all statement <u>except</u> those which contain the following verbs:

USE	ENTER	STOP
IF	GO	NOTE
EXIT	FIND	

<u>Examples:</u>

a. MOVE PAY TO AMOUNT; THEN GO TO 14.

b. CLOSE MASTER-FILE; THEN GO TO 63.

c. ADD TAX AND SALE; THEN GO TO 114.

d. READ MASTER, AT END GO TO 68; THEN GO TO 46.

e. WRITE ERROR; THEN GO TO 15.

IF STATEMENTS

The IF statement is designed to perform the following functions:

OPTION 1 - To determine if a certain condition(s) exists.

OPTION 2 - To compare the value of one field to another.

OPTION 3 - To determine the status of a field.

A GO TO clause is associated with each IF statement to indicate the next logical statement to be performed when that condition exists.

OPTION 1 (CONDITION-NAMES, BREAKPOINT SETTINGS)

This option permits the user to test the "ON" status of a computer breakpoint switch, or to test a "condition-name" against its specified value. (See Condition-Names, page 28, and Special-Names, page 8.)

Format:

<u>IF</u> condition-name THEN <u>GO</u> <u>TO</u> line-number-1

$$\left[; \left\{ \begin{array}{l} \underline{\text{OTHERWISE}} \\ \underline{\text{ELSE}} \end{array} \right\} \underline{\text{GO}}\ \underline{\text{TO}}\ \text{line-number-2} \right] .$$

The user may substitute for "condition-name" a special-name assigned to a breakpoint switch, or a condition-name used in the DATA DIVISION.

Example A: (Assume Breakpoint-1 has been assigned the special-name, MONTHLY-SUMMARY.)

```
20. IF MONTHLY-SUMMARY GO TO 47.
21. MOVE .......... .
```

If Breakpoint #1 is "set" (ON) when this statement is executed, the program will transfer to line 47. If this switch is not set (OFF), statement 21 will be executed.

Example B: (Assume that the CODE field had been described as follows:)

```
2 CODE; SIZE 1.
  88 DEPOSIT; VALUE IS "A".
  88 WITHDRAWAL; VALUE IS "B".
```

```
3Ø. IF DEPOSIT GO TO 1Ø6.
4Ø. IF WITHDRAWAL GO TO 11Ø.
5Ø. MOVE .......... .
```

When statement 30 is performed, the Narrator will test the CODE field against the value "A". If an "A" appears in that field, control will be transferred to statement 106; if not, the next statement will be executed.

OPTION 2: (RELATIONS)

This option allows the programmer to compare the value in a data field (or literal) to another data-field (or literal).

Format:

```
IF  { literal-1   }  relation-1  { literal-2   }  THEN GO TO line-number-1
    { data-name-1 }              { data-name-2 }

[; IF relation-2 THEN GO TO line-number-2]

[  { IF relation-3  THEN }                      ]
[ ;{ OTHERWISE           }  GO TO line-number-3 ] .
[  { ELSE                }                      ]
```

The user may substitute for "relation" one of the following:

```
IS [NOT] EQUAL TO          IS UNEQUAL TO
IS [NOT] LESS THAN         EQUALS
IS [NOT] GREATER THAN      EXCEEDS
```

Example A: IF AMOUNT IS LESS THAN BALANCE GO TO 62.
(Note: program continues in sequence on an "equal" or "greater" condition.)

Example B: IF REORDER-QTY EXCEEDS ON-HAND GO TO 63; IF EQUAL TO GO TO 62.
(Note: program continues in sequence on a "less" than condition.)

Example C: IF DATE OF MASTER EQUALS TODAYS-DATE GO TO 41; OTHERWISE GO TO 76.
(Note: on a "less" or "greater" condition the program transfers to line 76.)

OPTION 3: (STATUS)

This option permits the programmer to test the status of a numeric or alphanumeric field. (This option cannot be used with an alphabetic field.)

Format:

```
IF data-name-1  IS test-1  THEN GO TO line-number-1

[; IF  test-2  THEN GO TO line-number-2]

[  { IF test-3  THEN }                      ]
[ ;{ OTHERWISE       }  GO TO line-number-3 ] .
[  { ELSE            }                      ]
```

One of the following may be substituted for "test" if the field is NUMERIC:

$\boxed{\text{NOT}}$ POSITIVE $\boxed{\text{NOT}}$ ZERO

$\boxed{\text{NOT}}$ NEGATIVE $\boxed{\text{NOT}}$ NUMERIC

If the field is ALPHANUMERIC, the programmer may only test for the following:

$\boxed{\text{NOT}}$ NUMERIC

Example A: IF BALANCE IS ZERO GO TO 14; IF NEGATIVE GO TO 43; IF POSITIVE GO TO 56.
(Note: program will transfer to the path corresponding to the first "satisfied" test.)

Example B: IF BALANCE IS ZERO GO TO 73; OTHERWISE GO TO 64.
(Note: program will transfer to line 64 on both "positive" and "negative" quantities.)

Example C: IF BALANCE IS NEGATIVE GO TO 64.
(Note: program continues in sequence on a "positive" or "zero" quantity.)

IMPORTANT COMMENTS REGARDING IF STATEMENTS

1. The CLASS of each term involved in the test or relationship must be compatible. The following combinations are illegal:

A field		B field
Alphabetic	⟷	numeric
Numeric (edited)	⟷	Numeric (non-edited)

2. When comparing relationships, the following rules must be observed:

   a) Both terms must be fixed in size. (Variable fields cannot appear in an IF statement.)

   b) Both terms must include the same number of DATA characters. (ISS symbols and signs are not considered as data characters.)

   c) If either term includes a sign, coding will be generated in the object program to interrogate the sign locations and to determine if negative values are present, before actually comparing the values in the fields themselves.

   If "negative" quantities will NOT appear in these signed field the minimum number of machine instructions will be generated if the programmer compares data only. He should, for example, define these fields as follows:

```
2 TOTAL; SIZE 6; CLASS NUMERIC; SIGNED.
  3 T-AMOUNT; SIZE 5.
  3 FILLER; SIZE 1.

2 AMOUNT; SIZE 6; CLASS NUMERIC; SIGNED.
  3 A-AMOUNT; SIZE 5.
  3 FILLER; SIZE 1.
```

   Comparing the level-3 entries would then produce the least amount of machine instructions:

310. IF T-AMOUNT EQUALS A-AMOUNT GO TO 517.

d) If the data description for a NUMERIC field indicates "editing," the USER CANNOT COMPARE A PURE NUMERIC VALUE TO THAT FIELD.

A numeric field is assumed to be "edited" if:

1. the PICTURE clause contains editing symbols ($, . Ø B).

2. one of the following clauses appear:

ZERO SUPPRESS
CHECK PROTECT
FLOAT DOLLAR SIGN
or
BLANK WHEN ZERO

<u>Example A</u>:

Assume that the field, TOTAL, had been defined as follows:

2 TOTAL; SIZE 7; CLASS NUMERIC; POINT LEFT 2; ZERO SUPPRESS.

When the value in this field is zero, the field will appear as follows:

•	-	-	-	-	Ø	Ø

↑assumed decimal point.

Thus, if the programmer wishes to check this field for a zero quantity, he <u>CANNOT</u> write:

IF TOTAL IS EQUAL TO ZERO GO TO 4Ø.

Instead, he must write the following statement:

IF TOTAL IS EQUAL TO "++++ØØ" GO TO 4Ø.

If the user wanted to compare this field for a value greater than "50":

IF TOTAL IS GREATER THAN "++5ØØØ" GO TO 4Ø.

<u>Example B</u>:

Assume that the field, TOTAL, had been defined as follows:

2 TOTAL; SIZE 8; CLASS NUMERIC; POINT LEFT 2; PICTURE IS K$999.99.

If a numeric value of "4411" had been moved to TOTAL, the field would appear as follows:

•	$	Ø	4	4	.	1	1

The following statement, then, would test this field for a value greater than $50:

```
IF TOTAL IS GREATER THAN "$Ø5Ø.ØØ" GO TO 4Ø.
```

e) Literals appearing in IF statements must agree with the format of the other operand in the statement. The user cannot, for instance, compare numeric literals to alphabetic fields; nor may he compare literals that are not equal in size to the contents of a data field.

f) If an alphanumeric field is compared to a signed numeric field, the user may only test for an equal or unequal condition. In this case, the sign location in the numeric field will be considered as a character of data, and not a sign. For example:

```
        FIELD-A (AN)     FIELD-B (NUMERIC, SIGNED)

          X X X X          X X X S

    IF FIELD-A IS EQUAL TO FIELD-B GO TO 4ØØ.
```

The coding generated in the object program for this statement will compare the four characters in field-A with the four characters in field-B.

OPTION 1

$$\underline{\text{MOVE}} \begin{Bmatrix} \text{data-name-1} \\ \text{literal} \end{Bmatrix} \underline{\text{TO}} \text{ data-name-2 } \left[\text{, data-name-3...etc...} \right] .$$

Function: To transfer data to one or more areas, performing any editing, padding, or truncation as required by the description of the receiving area(s).

I. GENERAL RULES

A. CLASS

In all movements of data, the class of the sending and receiving areas must conform to the following rules; otherwise, an error will occur:

SENDING AREA	RECEIVING AREA			
	ALPHA	AN	NUMERIC	NUMERIC(EDITED)
ALPHA	Yes	Yes	No	No
AN	Yes	Yes	Yes	Yes
NUMERIC	No	Yes	Yes	Yes
NUMERIC (EDITED)	No	Yes	No	Yes

B. SIZE

In all moves the "net" size of the sending area must be less than or equal to the "net" size of the receiving area.

"Net" size is determined by subtracting the following from the length specified in the SIZE clause:

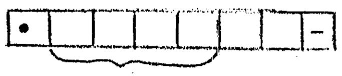
a) ISS symbol (if present)

b) sign (if present)

c) decimal locations (if present)

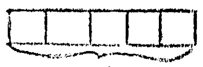
Examples:

2 BALANCE; SIZE 8; CLASS NUMERIC; POINT LEFT 2; SIGNED.



net size 4

2 CODE; SIZE 5; CLASS ALPHABETIC; PICTURE J.



net size 5

Note: If the receiving area is other than NUMERIC, signs and decimal locations will not be subtracted from the sizes of the sending and receiving areas.

C. JUSTIFICATION

When the net size of the receiving area is larger than the net size of the sending area, data will be "justified" in the receiving area according to the CLASS of the receiving area. The balance of the receiving area will then be "padded," i.e., filled with zeros or spaces.

The rules governing standard and non-standard justification are discussed on page 38, subject to the following exception:

> Only "standard" justification is permitted when either the sending or receiving area is variable.

Example A:

MOVE AMOUNT TO BALANCE.

AMOUNT

Ø	Ø	1	7	4	1

BALANCE

Before

3	3	7	1	2	2

After

Ø	Ø	1	7	4	1

Example B:

MOVE AMOUNT TO BALANCE.

AMOUNT

3	2	5	4	4

BALANCE

Before

1	4	3	2	6	6	7	7

After

Ø	Ø	Ø	3	2	5	4	4

Example C:

MOVE CITY TO LOCATION.

CITY

•	B	O	S	T	O	N

LOCATION

Before

•	C	L	E	V	E	L	A	N	D	-

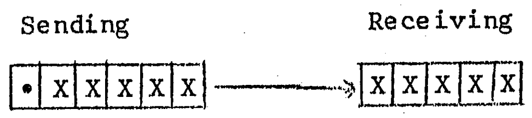
After

•	B	O	S	T	O	N	-	-	-	-

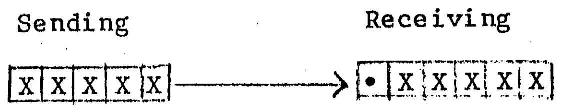
II FIXED FIELDS

A. ISS SYMBOLS

If the sending field contains an ISS, but the receiving field does not, the ISS is not moved:



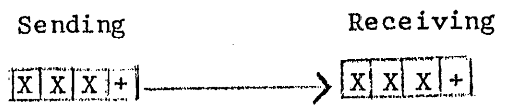
If the sending field does not contain an ISS symbol, but the receiving field does, an ISS will be generated in the receiving field.



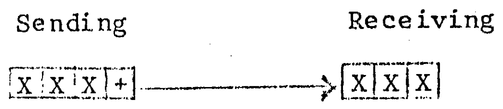
B. NUMERIC RECEIVING FIELDS

1. Signs

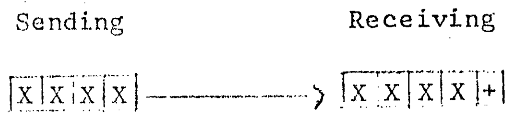
When both fields contain signs, the sign will be moved into the receiving field:



If the sending field contains a sign, and the receiving field does not, the sign will not be transferred:



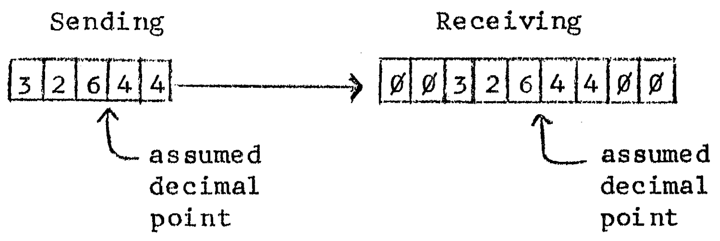
If the sending field does not contain a sign, but the receiving field does, a positive sign will be generated in the receiving field:



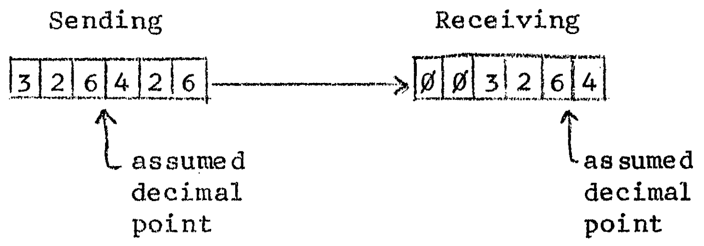
2. Decimal Locations

When numeric data is moved to NUMERIC receiving fields with standard justification (i.e., right-justified) decimal locations will be aligned with "truncation" or "zero-fill" in the receiving area, as required.

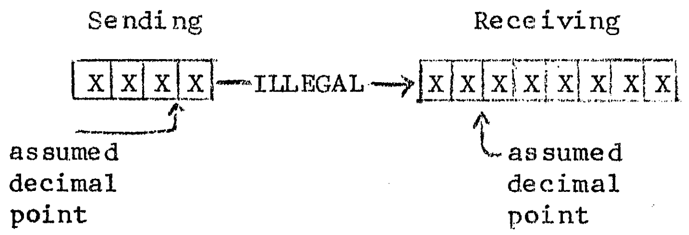
Example of "zero-fill"



Example of "truncation"

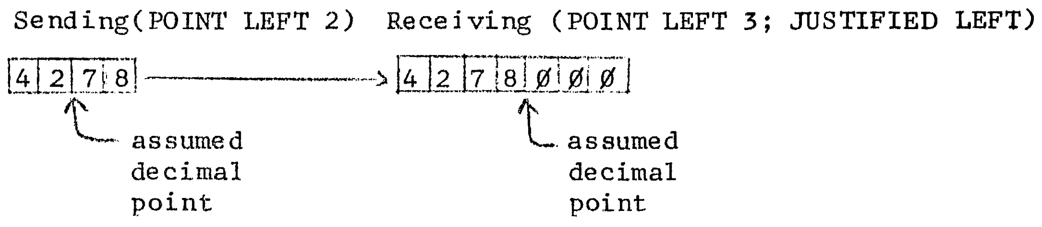


As indicated, data stored into the receiving field is governed by the specifications of the receiving field. It is possible, therefore, to truncate decimal locations that appeared in the sending field. It is an error, however, if the receiving field cannot accommodate all of the integral digits (not decimal locations) in the sending area. For example, the following is INCORRECT:



(Receiving field can only accommodate two integral digits.)

When numeric data is moved to NUMERIC receiving fields with non-standard justification (i.e., left-justified), decimal locations will not be aligned. For example:



3. Editing

"Editing" will be performed on numeric data moved into a numeric receiving area, if

a) RECEIVING field has been specified as ZERO SUPPRESS, FLOAT DOLLAR SIGN, CHECK PROTECT or BLANK ZERO.

Example: Assume that the clause "FLOAT DOLLAR SIGN" appeared in the description of the field, BALANCE.

144. MOVE DIFFERENCE TO BALANCE.

DIFFERENCE

Ø	Ø	Ø	1	4	9	5

BALANCE

Before	$	4	Ø	5	7	7	2	Ø
After	-	-	-	$	1	4	9	5

b) Receiving field description includes a PICTURE clause which contains editing symbols ($,.ØB):

Example: Assume that the field BALANCE was defined as follows:

2 BALANCE; SIZE 11; CLASS NUMERIC; PICTURE K$99,999.99; ZERO SUPPRESS.

144. MOVE DIFFERENCE TO BALANCE.

DIFFERENCE

•	Ø	2	1	4	Ø	6	2

BALANCE

Before	•	$	-	-	-	7	Ø	6	.	2	2
After	•	$	-	2	,	1	4	Ø	.	6	2

Note: Locations must be allowed in the size of the receiving field to accommodate the insertion of editing symbols.

C. ALPHABETIC OR AN FIELDS

When moving data to ALPHABETIC or AN receiving fields, a straight copy is performed, ignoring any editing, decimal point alignment or sign considerations. Unless specified otherwise, data will be left-justified with space fill.

Example:

Sending (NUMERIC; POINT LEFT 2; SIGNED)

Receiving (AN; POINT LEFT 2; SIGNED)

•	4	6	5	⊖	$\longrightarrow$	•	4	6	5	⊖	-	-	-	-

III VARIABLE FIELDS

A. REDUNDANT ZEROS AND SPACES

Redundant zeros or spaces will not be transferred into variable receiving fields. Redundant zeros appear to the left of the first non-zero character in NUMERIC fields; redundant spaces appear to the right of the last non-space character in ALPHABETIC or AN fields.

In cases where the sending field contains all redundant zeros or spaces, only an ISS symbol will be transferred.

B. ISS SYMBOLS

Variable fields must always contain ISS symbols. In addition, if the sending field is variable, the receiving field (regardless of whether it is fixed or variable) must also contain an ISS symbol.

C. SIGNS

When a sending field is signed, and the receiving field is not, the sign will not be transferred.

When a sending field is not signed, but the receiving field is, a positive sign will be generated in the receiving area.

D. DECIMAL LOCATIONS

Both fields must contain the same number of decimal locations.

E. EDITING

If the receiving field is variable, editing will not be performed.

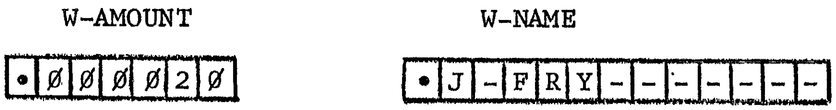
F. ASSEMBLING VARIABLE RECORDS VIA FIELD MOVES

Variable fields are located by means of their relative ISS position within the record. That is, the fourth field in a variable record would be accessable by locating the fourth ISS symbol starting from the beginning of the record.

If a variable record is being "assembled" by the user, all fields that include ISS symbols must be moved into the area IN THE SEQUENCE IN WHICH THEY APPEAR IN THE RECORD. If data for a particular field is not present, then the user must move an ISS into that field in order to maintain proper positioning.

When data is moved to a variable field, the Narrator places an ISS symbol immediately following the last data character. This permits the next variable field to be located. When the Narrator recognizes that this is the last variable field in the record, an End Message symbol is inserted in lieu of an Item Separator.

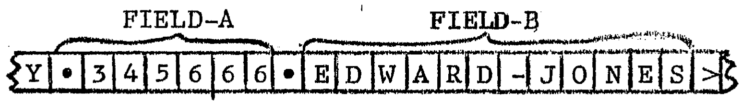
Example: Assume that W-AMOUNT and W-NAME appear as follows:



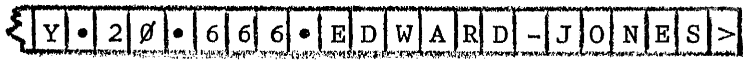
The user then moves these fields to the last two items in a variable record:

```
110. MOVE W-AMOUNT TO FIELD-A.
111. MOVE W-NAME TO FIELD-B.
```

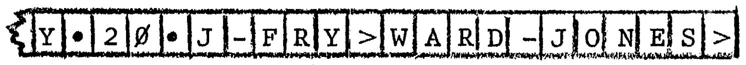
Record area Before Moves



Record area After First MOVE



Record area After Second MOVE



MOVING LITERALS

A. A literal may be moved into any data area other than a CONSTANT or another literal.

If the receiving area is fixed, literals will be moved according to the class of the receiving area (zero or space filling will occur, if necessary).

If the receiving area is variable, literals will not be expanded unless the FILLING option is specified. (Note that an ISS symbol will be added to the RHE of the literal to permit the Narrator to locate the next variable item.)

B. When moving literals to "edited" NUMERIC fields, the editing function will be ignored and the literal assumed to be in the exact format of the receiving area. (Exceptions will be ISS symbols and positive signs, which will be added to the literal, if required.)

Example: MOVE "$18.75" TO BOND-DEDUCTION.

C. When the receiving area format indicates an assumed decimal point, the literal is assumed to contain the proper number of decimal places.

Example: When moving eighteen dollars to a four character field with two decimal places, the literal should appear as "1800". The literal "18" would be assumed to be eighteen cents.

D. Since "unedited" NUMERIC fields cannot contain spaces, a literal moved to such a field cannot be the figurative constant SPACE(S), nor may spaces appear within the literal. (Note that a positive sign is not considered as a space.)

E. If a figurative constant, ALL any-literal or ALL figurative-constant is used, the FILLING option is implied. In these cases, the Narrator will generate a string of characters for the literal which will be the same size as the number of data positions in the receiving field.

Example: If MOVE ZEROS TO BALANCE is specified, and there are 8 data locations in the BALANCE field, a literal of eight zeros will be generated.

The generated literal will include an ISS and/or a positive sign if required in the receiving area. Decimal points, justification and editing will be ignored.

F. If a numeric literal is moved to a signed numeric field, a positive sign is not required within the literal. Negative signs, however, must appear.

literal — Receiving Field (SIGNED)

"467" $\longrightarrow \boxed{4|6|7|+}$

literal — Receiving Field (SIGNED)

"467⊖" $\longrightarrow \boxed{4|6|7|\ominus}$

MOVING CONSTANTS

In general, constants (level-77) are treated as normal data fields and, as such, follow the standard rules governing data field movement.

Constants, however, that are comprised of a single control symbol ("*") or contain more than one control symbol ("*AB*42"), are considered as SPECIAL constants. These constants are governed by the following rules:

1. If the constant is a single ISS (i.e., "*"), and it is moved to any data field (variable or fixed), only an ISS symbol will be moved. The balance of the data area will remain unchanged.

2. If more than one control symbol appears in the constant, the constant may only be moved to a variable area or to a fixed level-1 area. (If moved to a fixed level-1 area, the literal will be placed in the area according to the level-1 class. (Zero or space filling will occur if necessary.)

MOVE VERB

OPTION 2

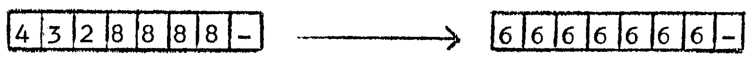
<u>MOVE</u> literal <u>FILLING</u> data-name-1 [, data-name-2...etc] .

This option permits the user to move a "literal" (or "figurative constant") to one or more destination areas. The value of the literal will be placed into the destination area starting at the left-hand end. If the receiving area is larger, the transfer will be repeated until the area is filled, at which time truncation occurs, if necessary.

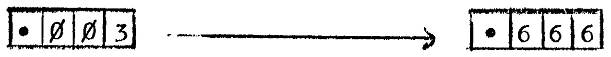
Example A

MOVE "6" FILLING AREA-1, AREA-2.

Area-1 before execution / Area-1 after execution



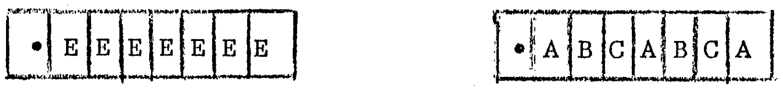
Area-2 before execution / Area-2 after execution



Example B

MOVE "ABC" FILLING AREA-1.

Area-1 before execution / Area-1 after execution



NOTES:

1. Literals are moved into data positions of receiving fields, only -- see example A.

2. If the receiving field is a signed numeric field, and the literal does not include a sign, the literal will be assumed to be positive, and a plus sign will be generated in the destination area.

3. When using the FILLING option, editing, justification, or alignment of decimal locations is ignored.

MOVING RECORDS TO RECORDS (Level-1)

When Both Records are Fixed

The following rules apply when both sending and receiving records are fixed:

1. The classes must be compatible.
2. The SIZE of each record must be the same.
3. The sending record is moved into the receiving area as a straight copy, ignoring all field descriptions for either record. Thus, the presence or absence of level-2 entries has no effect on fixed record moves.

Because individual fields are not considered in this type of MOVE, no editing, justification, decimal point alignment, truncation or padding will occur.

When Both Records are Variable

Same rules apply as when both records are fixed -- see above.

When One Record Is Fixed and Other Record is Variable

The following rules apply when one record is variable and the other record is fixed:

1. The classes must be compatible.
2. Both records must contain the same number of level-2 entries.
3. The corresponding "fixed portions" of each record must be identical, except for data-names.
4. The remaining portions of each record must be identical, except for the following:
   a) All fields (fixed and variable) must have ISS symbols.
   b) Receiving fields may be larger than sending fields.
   c) If a sending field is SIGNED, its corresponding receiving field must also be SIGNED. However, if the sending field is not SIGNED, the receiving field may or may not be SIGNED.
   d) Justification for all receiving fields must be standard.

SPECIAL MOVES

1. Moving Data Fields (Level-2, -3, and -77) To a Record

Data fields moved to a record area must be fixed, regardless of whether the level-1 area is fixed or variable. Data will be placed into the record area left-justified and the remainder of the record area, if any, will remain undisturbed. Note that if an ISS is associated with the sending field, it will be transferred.

If the record area is in message format, an End Message Symbol will be automatically inserted after the last character placed into the record area.

2. Moving a Record to a Data Field (Level-2, -3, or -77)

Records (variable or fixed) may be moved to a data field provided the data field is fixed. In moves of this type, data will be placed into the receiving field left-justified, regardless of the receiving field CLASS, and space filling will occur, if required.

An ISS location in the receiving field will be overlaid by the first character of the record area. Thus, it is assumed that the first character in the record area is an ISS Symbol.

MULTIPLY VERB

Function: To multiply two quantities together and store the result in the last name field or the specified one.

$$\underline{\text{MULTIPLY}} \quad \left\{ \begin{matrix} \text{literal-1} \\ \text{data-name-1} \end{matrix} \right\} \quad \underline{\text{BY}} \quad \left\{ \begin{matrix} \text{literal-2} \\ \text{data-name-2} \end{matrix} \right\}$$

$$\left[\underline{\text{GIVING}} \text{ data-name-3} \right] \quad \left[\underline{\text{ROUNDED}} \right] \quad \left[\text{; ON } \underline{\text{SIZE}} \ \underline{\text{ERROR}} \ \underline{\text{GO}} \ \underline{\text{TO}} \text{ line-number} \right] .$$

NOTES:

1. See RULES GOVERNING ARITHMETIC OPERATIONS, page 66.

   Note that the integer locations in the result field should be equal to the sum of the integer locations in the Multiplier and Multiplicand.

2. In addition to the above rules, the following applies:

   a) Literals used in the MULTIPLY statements must be integer values (whole numbers).

EXAMPLES:

17. MULTIPLY SALE BY STATE-TAX GIVING TAX-AMT ROUNDED.
211. MULTIPLY "2" BY AMOUNT.
423. MULTIPLY DEPENDENTS BY "6ØØ" GIVING DEDUCTION.

NOTE VERB

Function: To allow the user to include explanatory comments in his program.

NOTE ....

Example:

```
120. WRITE NEW-MASTER.
130. NOTE SPECIAL REPORT LISTING ROUTINE FOR CUSTOMER ACCOUNTS THAT
     HAVE HAD NO ACTIVITY DURING PAST YEAR.
140. MOVE AMOUNT TO BALANCE.
150. MOVE ACCOUNT-NO TO REPORT-ACCOUNT.
...
...
```

NOTE statements are for programmer use only; they have no effect on the object program. In the above example, then, statement 140 will be executed in the object program immediately following statement 120.

All NOTE statements must be assigned a specific line-number, and may not exceed 300 characters which includes the word NOTE and spaces.

NOTE statements cannot be referenced within the program.

OPEN VERB

Function: To initiate the processing of both input and output files and to Perform the checking or writing of tape labels.

```
OPEN   [ INPUT file-name-1        [, file-name-2 ...etc...] ]
       [; OUTPUT file-name-3      [, file-name-4 ...etc...] ]  .
```

When an OPEN statement is executed for any file, the tape label procedures are instituted, if required. See Tape Labels, page 126. The file is then positioned to release or to receive the first data record.

NOTES

1. The OPEN statement appears in the program at the point where the file(s) would logically be opened. It should be remembered that the programmer must indicate in this statement which files are INPUT and which files are OUTPUT.

2. The OPEN verb does not affect records that appear in the file. File records may only be obtained or released by appropriate READ and WRITE statements. Also, an OPEN statement must be executed prior to any READ or WRITE command for that file.

3. A second OPEN of a file cannot be executed until a CLOSE of that file has been performed.

4. If an OPEN statement refers to an input file that has been designated as OPTIONAL (See page 9), the Input/Output Control will request this information at object program running time. (This procedure is explained in a separate publication.)
   If the file is not present, it will not be "opened" and a print-out indicating the absence of the file will occur. When the first READ for this file is encountered, control will be transferred to the line-number specified in the AT END GO TO associated with the READ statement.

5. When tape files are "opened" they are assumed to be properly positioned, as they are not automatically rewound to BTC.

6. No label procedures are performed for files assigned to the PRINTER.

EXAMPLES

```
OPEN INPUT MASTER-FILE.
OPEN OUTPUT ERROR-LISTING.
OPEN INPUT MASTER-FILE, TRANSACTIONS-FILE.
OPEN INPUT MASTER-FILE, TRANSACTIONS-FILE; OUTPUT NEW-MASTER-FILE, DELETIONS,
  ADDITIONS.
```

PERFORM VERB

Function: To permit the execution of one statement (or a group of statements) a specified number of times, and to return control to the next statement in sequence.

$$\underline{\text{PERFORM}}\ \text{line-number-1}\ \left[\begin{Bmatrix}\underline{\text{THROUGH}}\\ \underline{\text{THRU}}\end{Bmatrix}\ \text{line-number-2}\right]\ \left[\underline{\text{EXACTLY}}\ \text{integer}\ \underline{\text{TIMES}}\right].$$

During the data processing operation, there will be many occasions where the programmer will be duplicating a statement, or a group of statement, that appear elsewhere in the program. Instead of writing these statements a second time (thereby generating the same machine instructions in the object program) the programmer may use the PERFORM verb.

HOW THE PERFORM OPERATES

When a PERFORM statement is encountered during compilation, the Narrator will generate in the object program a "BY-PASS" GO TO instruction following the last statement in the perform range. This BY-PASS will be initially set to the next statement in sequence. Thus, if the statements "PERFORM 5." and "PERFORM 80 THRU 86." appear in the program, the Narrator will generate two BY-PASS instructions -- one following statement 5 and another after statement 86.

Example A

```
  1. READ
  2. MOVE
  3. ADD
  4. WRITE
  5. READ
  6. MOVE
 ┌- - - - -┐
 ¦GO TO 7  ¦<-------- BY-PASS
 └- - - - -┘
  7. SUBTRACT
  8. WRITE
 ...
 ...
 ...
 25. PERFORM 3 THRU 6.
 26. MOVE
```

When a PERFORM statement is executed (for example, at line 25 above), the following events occur:

1. The line-number of the statement following the PERFORM is placed into the BY-PASS. In example A, then, the by-pass would be "set" to line 26.

2. Control is transferred to the first statement in the range (statement 3).

3. At the completion of the range, the by-pass is automatically reset to its initial line number (i.e., line 7) and control is transferred to line 26.

(Cont'd)

EXACTLY integer TIMES

When this option is specified, the Narrator establishes an internal counter that is checked each time the range has been performed. This counter is reduced by "one" after each performance, and control is returned to the first statement in the range. When the counter is recognized as "zero," the BY-PASS is reset to its initial line-number, and control is transferred to the statement following the PERFORM.

"Integer" may range from 1 to 4095.

IMPROPER USAGE OF THE PERFORM VERB

The following conditions must be avoided when using the PERFORM verb:

A. Last Statement In Range Is A Branch Statement (GO TO, IF).

Example B

```
 1. READ
 2. MOVE
 3. ADD
 4. WRITE
 5. READ
 6. GO TO 72.
    ┌─────────┐
    │GO TO 7  │<──── BY-PASS
    └─────────┘
 7. SUBTRACT
 8. WRITE
...
...
...
66. PERFORM 3 THRU 6.
67. MOVE
```

If we look at this example, we see that the last statement in the range (line 6) is a GO TO statement. Thus, if any attempt is made to perform this range, control will always be transferred at line 6 to line 72, AND THE BY-PASS WILL NEVER BE EXECUTED. In this case, then, a return to the statement following the PERFORM can never be accomplished.

B. Transferring Out Of The Perform Range

The BY-PASS is only reset (i.e., returned to its original line number) after the last statement in the range has been executed. Therefore, if control is transferred outside the range, and never returned back into the range, the BY-PASS will not be reset. Should the user subsequently execute this part of his program in sequence, the BY-PASS will NOT be set to the next sequential statement, but rather to the line number following the last PERFORM that affected that range.

(Cont'd)

C. Improper Nesting

"Nesting" (i.e., PERFORM statements appearing within a PERFORM range) is permitted, provided there is no overlapping. The following examples illustrate both proper and improper nesting.

Example C: Correct Nesting

```
 2.
 3. PERFORM 82 THRU 89.
 4.
 5.
 6.
 7.
 8.
 9.
10.
11.
12.
...
...
83. PERFORM 5 THRU 7.
...
...
94. PERFORM 2 THRU 1Ø.
95.
```

Example D: Incorrect Nesting

```
 2.
 3.
 4. PERFORM 8 THRU 12.
 5.
 6.
 7.
 8.
 9.
1Ø.  <-------------  BY-PASS
11.
12.
13.  <-------------  BY-PASS
...
...
...
...
67. PERFORM 2 THRU 1Ø.
68.
```

To understand the problem of incorrect nesting (as illustrated in example D) let us assume that the program is now at statement 67. Since this is a PERFORM statement, the BY-PASS after line 10 will be "set" to return to line 68 before control is transferred to line 2.

While performing this range of statements, another PERFORM is encountered at line 4 in which the range 8 through 12 is to be executed. If we follow the program from this point we see that the BY-PASS following line 10 will transfer out to line 68. Thus, improper overlapping has prevented the execution of the statements at lines 11 and 12.

USING THE EXIT VERB

If, while executing statements within a PERFORM range, it is desired to exit from the range, a COMMON exit point must be provided. This is accomplished by using the EXIT verb as the last statement in the range. For example:

```
           2.
           3.
PERFORM    4. IF NEGATIVE GO TO 7.
  RANGE    5.
           6.
           7. EXIT.
              GO TO 8  <------ BY-PASS
           8.
           9.
```

Note that the EXIT verb does not produce additional coding in the object program.

GENERAL COMMENTS CONCERNING THE PERFORM VERB

1. A PERFORM statement cannot refer to line-numbers appearing in another SECTION -- see Segmentation, page 118.

2. There is no necessary relation between "line-number-1" and "line-number-2," except that the sequence of steps starting at "line-number-1" must proceed logically to the statement at "line-number-2" in order to complete the range.

3. In general, "line-number-1" should not be the next statement after the PERFORM. If it is, the result will be that the loop will be completed one more time than was probably intended, because after the PERFORM has been satisfied, control will be returned to "line-number-1" and the program will continue from that point. In the following example, then, statement 22 THRU 26 would actually be performed <u>three</u> <u>times</u>.

```
21. PERFORM 22 THRU 26 EXACTLY 2 TIMES.
22.
23.
24.
25.
26.
```

EXAMPLES

```
PERFORM 5.
PERFORM 5 EXACTLY 2 TIMES; THEN GO TO 444.
PERFORM 2ØØ THROUGH 3Ø7.
PERFORM 2ØØ THROUGH 3Ø7 EXACTLY 1Ø TIMES.
```

READ VERB

Function: To make available the next "logical" record from an input file, and to permit the performance of an imperative statement when the end of file (EF symbol) is detected.

READ file-name RECORD [; AT END GO TO line-number] .

NOTES:

1. The READ statement applies to file-names, not record -names.

2. An OPEN statement for the file must be executed before the first READ command is given.

3. The READ command places the next "logical" record into the file area for processing. (If the file contains more than one type of record, it must be determined through programming what type record appears in the file area.)

4. Every READ statement must have an implicit or explicit AT END GO TO option.

   If the AT END GO TO options are identical for all reads of the same file, the programmer need only include this clause with one read statement. The compiler will then append this clause, with its associated line-number, to all other READ commands for that file. If different AT END GO TO statements apply to the same file, ALL reads for that file must have explicit AT END GO TO statements. (If no END statement exists for a file, the compiler will indicate an error.)

5. After executing an AT END GO TO statement, any attempt to perform another read from that file will constitute an error in the OBJECT program, unless the file had been closed and reopened again.

6. When the Input/Output Control recognizes that all records have been processed on an intermediate reel, the following occurs:
   a) The End-Label is checked (if present).
   b) The USE subroutine is performed (if present).
   c) A tape swap is effected if more than one tape station has been assigned to that file. If only one tape station was assigned, the program will stop and indicate that a new reel is to be mounted.
   d) The Beginning-Tape-Label of the next reel is checked (if present).
   e) The USE subroutine is performed (if present).
   f) The next "logical" record is made available.

7. When the Input/Output Control recognizes that all records have been processed on the last reel, the following occurs:
   a) The End-Label is checked (if present).
   b) The USE subroutine is performed (if present).
   c) Control is transferred to the AT END GO TO line-number associated with the last READ command.

8. If an OPTIONAL Input File is not present at object program running time, the first READ encountered for that file will cause a transfer to the "AT END GO TO" routine. Subsequent READ commands will cause an error print-out.

EXAMPLES

```
READ MASTER-FILE; AT END GO TO 4Ø7.
READ MASTER-FILE.
READ SALES-FILE; AT END GO TO 51Ø.
READ SALES-FILE; AT END GO TO 5ØØ.
```

STOP VERB

Function: To stop the computer when the program has been completed, or to cause a temporary halt during processing.

$$\underline{\text{STOP}} \left\{ \begin{matrix} \underline{\text{RUN}} \\ \text{literal} \end{matrix} \right\} .$$

Example A:

```
410. STOP RUN.
```

When the word RUN is specified, "END OF RUN" will be displayed on the monitor printer and the computer will stop. At this point the ending procedure established by the installation should be instituted.

If the object program will be used with the RCA Program Sequencer, the next program will be processed when the STOP RUN statement is executed.

Example B:

```
540. STOP "INVALID TRANSACTION ACCOUNT NUMBER".
```

When a literal is specified, the literal will be displayed on the monitor printer when the program stops. Depressing the start button at this point will cause the execution of the next sequential statement.

SUBTRACT VERB

Function: To subtract one quantity (or a sum of quantities) from a specified quantity, and to store the result in the last named field or in the specified one.

```
                                 ┌                                  ┐
            ┌ literal-1     ┐    │ ┌ ,   ┐ ┌ literal-2     ┐        │      ┌ literal-n     ┐
SUBTRACT    │               │    │ │     │ │               │ etc... │ FROM │               │
            └ data-name-1   ┘    │ └ AND ┘ └ data-name-2   ┘        │      └ data-name-n   ┘
                                 └                                  ┘

[GIVING data-name-m] [ROUNDED] [; ON SIZE ERROR GO TO line-number] .
```

NOTES:

1. See RULES GOVERNING ARITHMETIC OPERATIONS, page 66. Note that when the integer locations of the result field equals the integer locations of the largest operand, a "possible" overflow condition may exist.
2. In addition to the above rules, the following apply:

   a) A maximum of 10 operands can be specified in any single SUBTRACT statement. This includes the GIVING field, if present.

   b) Literals must contain the same number of decimal locations as the result field. If, for example, the literal "5ØØ" has been used, and the result field has two decimal locations, the literal is considered as being $5_\wedge ØØ$. If the result field has no decimal locations, the literal is considered to be $5ØØ_\wedge$.

   c) When dealing with multiple subtrahends, the effect is the same as if all subtrahends were first summed, and this sum then subtracted from the minuend. Note that minuend is literal-n or data-name-n.

EXAMPLES

```
 17.SUBTRACT DEDUCTIONS FROM PAY.
129.SUBTRACT DEDUCTIONS FROM PAY GIVING NET-PAY.
473.SUBTRACT AMT-1, AMT-2, AMT-3, AMT-4, AMT-5 FROM BALANCE.
```

USE VERB

Function: To designate special procedures that are to be performed when a tape label is checked or created.

```
USE line-number-1  [ {THROUGH} ]  line-number-2     AFTER STANDARD
                   [ {THRU   } ]

[ {BEGINNING} ]                              {file-name}
[ {         } ]  REEL LABEL PROCEDURE ON     {INPUT    }   .
[ {ENDING   } ]                              {OUTPUT   }
```

When a USE subroutine has been specified for a file, the statement(s) in that routine will be performed immediately FOLLOWING the tape label routine. In the case of an Input file, the following sequence of events takes place:

1. Label is read.
2. The "Standard" label items are checked -- see Tape Labels, page 126.
3. USE subroutine is performed.
4. Program continues in sequence.

When an Output file is involved, the procedure is as follows:

1. The "Standard" label items are placed in the label area -- see Tape Labels, page 126.
2. USE subroutine is performed.
3. The tape label is written to the file.
4. Program continues in sequence.

NOTES

1. The line numbers specified within the USE statement MUST be greater than the line number in which the USE verb appears. For example, the USE statement at line 307 cannot contain references to statements whose line numbers are less than 307.

2. During compilation, all USE statements and their associated sub-routines are REMOVED from the normal statement sequence. These routines are executed, as required, by the Input/Output Control.

   Therefore, when USE statements and USE subroutines appear within the program, they must be considered as "outside" the normal

112. USE 113 THRU 114 AFTER STANDARD BEGINNING REEL LABEL PROCEDURE ON NEW-MASTER-FILE.

113. MOVE "B" TO PRIORITY-CODE.

114. MOVE ACCOUNT-NUMBER OF MASTER TO FIRST-MSG-CRITERIA.

Example B Assume that a program has three files: MASTER-FILE (input), ERROR-FILE (output), and NEW-MASTER-FILE (output). It is desired to cross-foot the record counters; i.e., the record count for the ERROR and NEW-MASTER files are to be added and compared to the record count of the input file during the end-of-run procedures.

62. USE 63 AFTER STANDARD ENDING REEL LABEL PROCEDURE ON ERROR-FILE.

63. ADD BLOCK-COUNT AND CHECK-AREA-A.

64. USE 65 AFTER STANDARD ENDING REEL LABEL PROCEDURE ON NEW-MASTER-FILE.

65. ADD BLOCK-COUNT AND CHECK-AREA-B.

66. USE 67 AFTER STANDARD ENDING REEL LABEL PROCEDURE ON MASTER-FILE.

67. ADD BLOCK-COUNT AND CHECK-AREA-C.

(Note: The BLOCK-COUNT in the ending label specifies the number of "physical" records on that reel.)

processing path. For example:

```
     As Originally Written          As Executed in Program

400. READ ......                    400. READ....
410. USE 450 THRU 490...            430. MOVE....
430. MOVE.....                      440. ADD.....
440. ADD......                      530. MOVE....
450.
 .
 .
 .
490.
500. USE 510 THRU 520...
510.
520.
530. MOVE.....
```

3. The statement containing the USE verb cannot be referenced by any other statement.

4. USE subroutines are performed during label processing by the Input/Output Control. Entering these subroutines from elsewhere in the program, therefore, may cause an error at object program running time. Similarly, transferring from the subroutine to another part of the program may have the same effect.

   An error warning will be given at compilation time if any such transfers of control have been made by the programmer.

5. Input-Output statements may not appear in a USE subroutine except for the DISPLAY and ACCEPT verbs. SECTION and other USE verbs, of course, cannot appear within a USE subroutine.

6. The Label area can only be accessed through a USE subroutine at the time label checking is performed. (Note that label checking occurs not only when files are opened or closed, but also when reels are exhausted or replenished.)

7. If BEGINNING/ENDING is omitted, the designated procedures will be executed for both beginning and ending labels.

   If INPUT is specified, the designated USE subroutine will be applied to all input files and an error will occur if another USE subroutine appears for any input file. The same applies to the use of OUTPUT.

<u>Example A</u> The beginning-tape label for the NEW-MASTER-FILE appears as follows:

< • ID•Date-Written • Priority-Code • First-Msg-Criteria >

Since the Priority-Code and First-msg-Criteria are <u>non-standard</u> label items, the user must move this information into the label area through a USE subroutine. (Note that <u>standard</u> label items are automatically placed in the label area.)

WRITE VERB

Function: To release a "logical" record for an Output file, and to permit vertical positioning if the Output file has been assigned to the On-Line Printer.

```
              WRITE record-name     [FROM data-name]

        [ {BEFORE}                   {integer LINES} ]
        [ {      }    ADVANCING      {PAGE         } ] .
        [ {AFTER }                   {LOOP         } ]
```

"Record-name"

"Record-name" must be an output record; it cannot be an input or working-storage record.

Data For "record-name" In Output File Area

If the information for the "record-name" appears in the output file area, the user need only state:

```
WRITE record-name.
```

Data For "record-name" Not In Output File Area

When the information for the "record-name" is not in the output file area, the user must either move the information into the output area or state:

```
WRITE record-name FROM data-name.
```

In this case, the Narrator first "moves" the "data-name" to the "record-name," and then executes the WRITE. This performs the same function, then, as:

```
MOVE data-name TO record-name.
WRITE record-name.
```

"Data-name"

"Data-name" may be any level-1, 2, 3 or 77 area. However, the "data-name" cannot be qualified or subscripted.

If the format of "data-name" differs from "record-name," the "data-name" area will be moved into the output area according to the standard rules governing MOVE operations -- see MOVE, page 90.

ADVANCING OPTION

The ADVANCING clause MUST be used in all WRITE statements that pertain to files assigned to PRINTER.

This clause is used to specify that the paper is to be advanced a certain number of lines, the page is to be changed, or that vertical tabulation is to be controlled by the paper-tape-loop in the printer, itself.

As indicated, the usermust state whether the paper is to be advanced before or after printing.

For example, it is desired to write the Report record, then advance the paper 3 lines:

```
WRITE REPORT BEFORE ADVANCING 3 LINES.
```

Paper Advancing In Simultaneous Mode

Paper advancing will only be performed in the Simultaneous mode when the WRITE statement contains the BEFORE option. Simultaneity during printing will not occur if the AFTER option is used.

NOTES

1. An OPEN statement must be executed prior to the first WRITE for any file.

2. End of Reel:

   If the end of reel is recognized when a WRITE command is given, the following occurs:

   a) The end label procedures (if required) are instituted and the Ending-Tape-label is written to the file. See Tape Labels, page 126.

   b) A tape swap is performed if more than one tape station has been assigned to the file. Where only one tape station has been assigned, the program will stop and indicate that a blank reel is to be mounted.

   c) The beginning label procedures (if required) are instituted for the new reel and the Beginning-Tape-label is written. See Tape Labels, page 126.

   d) The "record-name" is written to the new reel.

   e) The program continues in sequence.

ON-LINE PRINTING

When On-Line printing is required, the Narrator will reserve a computer area of 120 characters. This area will be a "common" area that will be used for all files assigned to PRINTER. Thus, only one print file may be "opened" at any one time.

At the start of the object program this common file area is automatically cleared to spaces. (This is the only time that clearing of the print area is automatic.)

Because print files may contain different types of records, it cannot be assumed when developing a record in the print area that information from the previous record has been cleared to spaces. If data from the previous record has been completely "overlaid," clearing, of course, is not necessary.

In defining records appearing within print files, the SIZE of the record must be specified as 120 characters if the record is in "block" format, and 118 characters if in "message" format.

EXAMPLES

```
WRITE NEW-MASTER.
WRITE NEW-MASTER FROM WORK-MASTER.
WRITE REORDER BEFORE ADVANCING 4 LINES.
WRITE REORDER BEFORE ADVANCING PAGE.
```

SAMPLE OF PROCEDURE DIVISION

```
PROCEDURE DIVISION.

     1Ø. OPEN INPUT MASTER-FILE, TRANSACTION-FILE;
               OUTPUT NEW-MASTER-FILE, REORDER-FILE.

     2Ø. MOVE SPACES TO PRINT.

     3Ø. READ TRANSACTION-FILE; AT END GO TO 14Ø.

     4Ø. READ MASTER-FILE; AT END GO TO 16Ø.

     5Ø. IF PART-NUMBER OF MASTER IS LESS THAN PART-NUMBER OF
           TRANSACTION GO TO 11Ø, IF GREATER GO TO 12Ø.

     6Ø. MOVE MASTER TO REORDER.

     7Ø. SUBTRACT QUANTITY-USED FROM BALANCE ROUNDED.

     8Ø. IF  BALANCE IS GREATER THAN LEVEL GO TO 1ØØ.

     9Ø. WRITE REORDER BEFORE ADVANCING 2 LINES.

    1ØØ. WRITE NEW-MASTER FROM REORDER; THEN GO TO 3Ø.

    11Ø. WRITE NEW-MASTER FROM MASTER; THEN GO TO 4Ø.

    12Ø. DISPLAY "INVALID TRANSACTION" UPON MONITOR.

    13Ø. READ TRANSACTION-FILE; AT END GO TO 14Ø; THEN GO TO 5Ø.

    14Ø. READ MASTER-FILE; AT END GO TO 17Ø.

    15Ø. WRITE NEW-MASTER FROM MASTER; THEN GO TO 14Ø.

    16Ø. DISPLAY "MASTER EXHAUSTED" UPON MONITOR.

    17Ø. CLOSE MASTER-FILE, TRANSACTION-FILE, NEW-MASTER-FILE,
           REORDER-FILE.

    18Ø. STOP RUN.
```

EF
ED

SEGMENTING PROGRAMS

If the available computer memory cannot accommodate the object program, the program must be broken up into segments by the user. This is accomplished by PRECEDING each segment with the following statement:

line-number SECTION.

Example:

```
                 1. SECTION.
                ⎧2.
    Segment 1   ⎨3.
                ⎪4.
                ⎩5.
                 6. SECTION.
                ⎧7.
    Segment 2   ⎨8.
                ⎪9.
                ⎩10.
```

There may be up to twenty-one (21) segments in a Narrator program; if segmentation is not desired, SECTION statements are not to appear.

At program running time, segments will be brought into memory automatically, as required. For example, if a GO TO statement in segment "1" refers to a line-number in segment "2", segment 2 will be brought into memory and control transferred to the line-number specified. Also, after the last statement in a section is executed, the following section will be automatically entered. (Assuming, of course, that this last statement does not transfer to another area of the program.)

Programming Considerations When Using Segmentation

1. When a program is segmented the first line-number must be a SECTION statement.

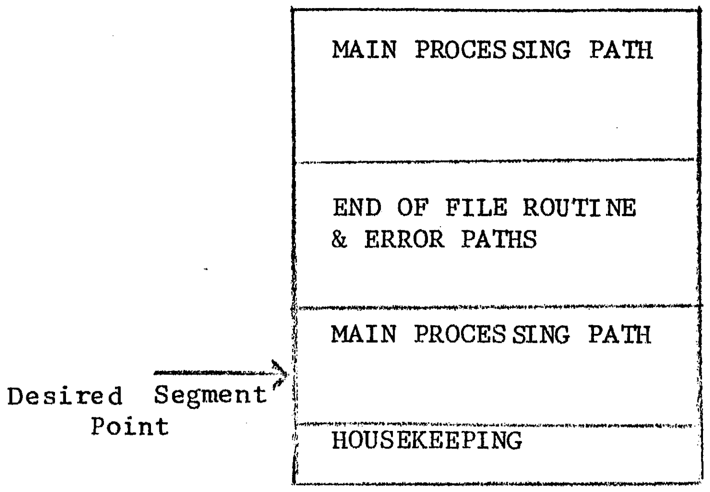
2. Only one segment appears in memory at any given time. Because of this, ALTER or PERFORM statements in one section cannot refer to line numbers that appear in another section.

3. Care should be exercised when defining the limits of a segment. For example, segments cannot end <u>within</u> a PERFORM RANGE, since the exit by-pass for that range appears in another segment and, consequently, cannot be properly set. Also, segments are called into memory in their original form, and not as modified by the program. This means that GO TO statements used as program "switches" will always appear as they were originally written whenever the segment is brought into the computer.

GENERAL COMMENTS

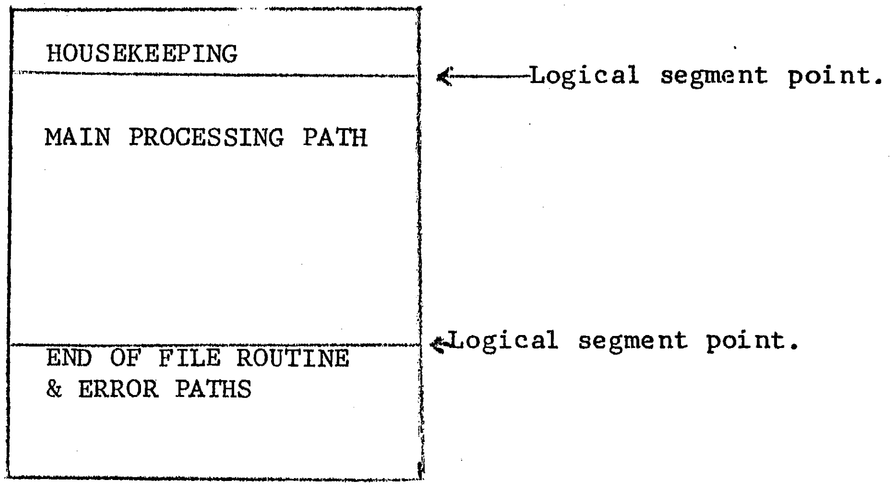
As a general rule, it is advisable for the programmer to keep the idea of segmentation "in mind," even though the program need not be segmented. If the user takes this approach -- and it later becomes necessary or desirable to segment -- the segmentation may be accomplished with a minimum amount of program modification.

If, for example, the program originally had been written as shown below, a good deal of modification might be required if the user wanted to segment as indicated.



Should the desired segment point fall within an "often-recurring" portion of the main processing path, a prohibitive amount of tape movement will occur at program running time to continually bring segments into the computer. In this case, the programmer may find it necessary to reorganize his program, requiring extensive changes to line numbers, GO TO addresses, IF statements, etc.

A problem such as this would have been avoided if the program had been organized as follows:



In this illustration, segmentation at "logical" segment points could have been accomplished without changing statements in the main processing path, or creating excessive tape movement at running time.

OWN CODE

The ability to insert 501 Assembly pseudo-code directly into the Narrator sequence is accommodated by the ENTER verb. Due to the difference in format between Narrator statements and Assembly pseudo-code, the pseudo-code instructions do not follow the ENTER verb. They are written on regular Assembly program sheets and are included after the PROCEDURE DIVISION. (The Assembly coding will be inserted during compilation at the place where the ENTER statement appears.)

The format for writing pseudo-code is described in the RCA AUTOMATIC ASSEMBLY SYSTEM manual, subject to the following rules:

FORMAT OF OWN-CODE

The Own-Code section may contain a number of independent "groups" of pseudo instructions. A "group" is defined as those pseudo instructions that pertain to a specific ENTER statement. The user must indicate the conclusion of each group by inserting "END" in the OP column of the line following the last instruction in the group. The entire OWN- CODE section is terminated by an "ED" symbol.

Example:

```
PROCEDURE DIVISION.
1. MOVE.....
.
.
.
468. DISPLAY....

          EF
```

PAA1Ø		DA	XXX	XXX
		LQ	XXX	XXX
		IT	XXX	XXX
		OCT	XXX	XXX
		END		
PAB1Ø		OCT	XXX	XXX
		END		
PAC1Ø		OCT	XXX	XXX
		OCT	XXX	XXX
PAC11		IT	XXX	XXX
		END		

```
     ED
```

P-ADDRESSES

1. The first instruction in each Own-Code group must have an explicit P-address. (This address must be identical to that appearing in the ENTER statement.)

2. There must be at least one explicit P-address for every 30 pseudo-instructions within each group. Explicit decimal point insertion P-addresses may appear in the instruction number column; relative P-addresses cannot.

3. Each Own-Code group may use P-addresses ranging from PXXØØ to PXX98, with the only restriction being that the second and third characters (i.e., XX) must be the same for all instructions within each "group." For example:

Group 1	Group 2
PAAØØ	PBB1Ø
-----	-----
-----	PBB22
PAA1Ø	-----
-----	-----
-----	-----
PAA6Ø	PBB25
-----	PBB22+1.1

REFERENCING OWN-CODING

1) The Narrator language can only refer to own-coding through the ENTER verb, i.e.:

```
12. MOVE PAY TO AMOUNT.
13. ENTER ASSEMBLER PAA1Ø.
14. MOVE ON-HAND TO GROSS.
-
-
-
68. GO TO 13.
```

2) Instructions in the Own-Code section may directly refer to Narrator line-numbers, provided the line-number referenced appears in the same section as the Pseudo-Code instruction.

Example

Inst.No.	OP	A	B	T	IF	GO TO	IF	GO TO
PAB21	TC	126						
PAB22	SC	#14#	#14#	DAMT(R)	+	31	-	224

3) There is no communication between instructions within one Own-Code group and instructions that appear in another Own-Code group.

OWN-CODE REFERENCE TO NARRATOR DATA NAMES

The Own-Code section may directly refer to FAA (Fixed and always appearing) data-names thathave been defined in the DATA DIVISION, subject to the following rules:

1. A "D" must be prefixed to all data-names. For example:

Narrator Data-name	Own-Code Reference
DATE	DDATE
NET-PAY	DNET-PAY
ACCUMULATIVE-INCOME-FOR-YEAR	DACCUMULATIVE-INCOME-FOR-YEAR
MASTER	DMASTER
TRANSACTION	DTRANSACTION

2. All names must be addressed character relative in the following format:

Dname(L) or Dname(L+n)

Dname(R) or Dname(R-n)

Example :

OP	A	B	T
SC	DPAY(L)	DPAY(R)	"*ØØØ"
IT	DSTOCK-NO(R)	DPART-NO(R)	
LW	DTRANSACTION(L)	T77	
SCC	DCOUNTER(L)	DCOUNTER(L+118)	

If the data name reference does not include a character relative designation, the Own-Code Generator will indicate a possible error and will assign the RHE address for this field, regardless of the format of the pseudo instruction.

3. All Non-unique data names must be qualified in the following manner:

OP	A	B
IT	DSTOCK-NO(R) IN MASTER	DSTOCK-NO(R) IN ERROR
DA	DAMOUNT(R) OF MASTER	"*5ØØ"

4. Since the Narrator accepts data names up to 30 characters, the usual 19 character-per-column limit for the Assembly system is disregarded in the Own-Code section. If a data name is too large to fit conveniently into an address column, the pseudo-code writer may "squeeze" the name into a single line as shown below:

Example :

OP	A	B	T
DA	DSALARY(R)	DACCUMULATIVE- INCOME-FOR-YEAR(R)	

5. The user may refer to the FIRST variable item in a variable file area by name; however, only the left-hand-end of this field may be addressed, i.e., Dvariable (L) or Dvariable (L+n).

INPUT/OUTPUT INSTRUCTIONS

The processing of records in the object program is under the direction of the Input-Output Control, and the user must be careful NOT TO EXECUTE INSTRUCTIONS IN THE OWN-CODE SECTION THAT DISTURB THE POSITIONING OF ANY FILE. If this is attempted, an error may occur in the running program.

SIMULTANEITY

The simultaneous gate will always be OPEN when own-coding is performed. If the user desires the gate closed, he MUST REOPEN IT BEFORE LEAVING THE OWN CODE SECTION.

ADDRESS MODIFIERS

All address modifiers may be used in the Own-Code section. However, the contents of these modifiers (except AM1) will change when control is transferred from own-coding back to the Narrator.

WORKING-STORAGE AREAS

Working-storage areas in the DATA DIVISION may only be addressed by prefixing a "D" to the data-name.

If the user desires to define working storage areas in the OWN-CODE section, he may only define areas in the W9 working storage "block."

Format:

W9xxxxx.n

where "n" may be up to four decimal characters.

Example:

OP	A	B	T
SCC	W9A.1Ø(CØ)	W9A	
STC	DAMOUNT(L)	DAMOUNT(R)	W9A

LIMITATIONS ON USE OF 501 AUTOMATIC ASSEMBLY CODING

The following limitations are imposed on the use of pseudo-coding in the Own-Code section:

1. Tape station addresses must be designated as Txx. I and O symbolics cannot be used.
2. The following mnemonic operation codes cannot appear in the Own-Code section:

```
   ADV
   DUP
   RES
   SGMT
  *ST
 **RD
 **MSW
```

*Although the mnemonic ST instruction may not be used, G76 or M76 operation codes may appear.

**The Own-Code section cannot create L-list names; this imposes a restriction then on the use of the mnemonic RD and MSW instructions. If Random Distribute or Multiple Sector Write instructions are desired, they must be written using G or M operation codes. (The list of addresses must be created in the A and B columns of the pseudo instructions.)

3. Descriptor Verbs

   Descriptor verbs cannot be used.

4. Macros and Subroutines

   Macros and subroutines cannot be used.

CONSTANTS

The following rules apply to the use of octal and decimal literals in the Own-Code section:

1. FIXED literals CANNOT be used.
2. K-constants CANNOT be used.
3. Any literal except three quotes (" " ") may be used.
4. If uniqueness is desired, U1 thru U199 must be specified.
5. As in normal Assembly operation, octal literals must contain an even number of characters, ranging from 0 - 7. If the Own-Code Generator recognizes an uneven count it will add a zero to the LHE of the literal and continue processing. An error will also be indicated at compilation time.
6. Literals used in the Own-Code section are not restricted to 19 characters and may range up to 120 characters. For example:

Inst. No.	OP	A	B
PAR21	LW	"IMPROPER+EMPLOYEE+CODE +FOR+FOLLOWING+RECORD/"	T77

Note that if the literal exceeds 19 characters, a slant symbol as the first character will be converted to an SM symbol, a slant as the last character of the literal will be converted to an EM symbol. THIS WILL NOT OCCUR IN LITERALS WHICH ARE 19 CHARACTERS OR LESS.

SIZE OF OWN-CODE MESSAGES

The physical size of an own-code message cannot exceed 400 characters (including a start and end message symbol).

EXAMPLE OF OWN-CODING

EF

Inst.No.	Comments	OP	A	N_A	N_B	B	T
PAAØØ		STC	#ØØ17#			#ØØ17#	DSELECT-PRINT(R)
		END					
PABØØ		STC	#ØØØØ#			#ØØØØ#	DSELECT-PRINT(R)
		END					
PACØØ	TO REWIND ALL TAPES	TCT	#ØØØØ75#(C1)			PACØ3(A)	
PACØ1	TO BTC AT START	OCT	PACØ3(3)			PACØ2(5)	
PACØ2	OF PROGRAM	TS	PACØ3+1			TØØ	N
		OCT	PACØ3(3)				
PACØ3	TALLY COUNTER	RWD					
		TA	PACØ1			PACØ3(A)	
		END					

ED

(Note: Although the Comments column may be freely used, comments will not appear in the object program listing. They will appear, however, in the source program listing.)

TAPE LABELS

The 501 NARRATOR System has been designed to provide automatic tape label checking and creation. In this respect the NARRATOR user has the following options:

1. to omit label-checking
2. to use RCA Standard Labels
3. to supply his own labels

1. LABELS OMITTED

When tape labels will not be used for a particular file, the File Label entry is written as follows: "LABEL RECORDS ARE OMITTED."

Files that do not contain tape labels appear in the following format:

SINGLE-REEL FILES

```
| EF ------------------- data ------------------- EF EF ED |
```

MULTIPLE-REEL FILES

```
| EF ------------------- data ------------------- EF ED |  <--Intermediate
                                                               reels

| EF ------------------- data ---------------- EF EF ED |  <--Final
                                                            reel
```

2. STANDARD LABELS USED

The Narrator user has the option of utilizing the RCA Standard Tape Label generation and label-checking features. In this case, the user does not define tape labels himself, but merely specifies in the File Description entry that "LABEL RECORDS ARE STANDARD".

The Narrator, upon recognizing this clause, will:

1. Generate a standard Beginning and End Tape label for this file.
2. Write these labels on the file when required.
3. Automatically check these labels at the time the file is opened, when reels are exhausted, and when the file is closed.

Files that contain STANDARD tape labels appear in the following format:

SINGLE-REEL FILES

```
| < BTL >  EF ------------- data ------------- EF < ETL > EF ED |
```

MULTIPLE-REEL FILES

```
[< BTL >  EF ————————————— data ————————————— EF   < ETL >   ED ]←Intermediate
                                                                   reels

[< BTL >  EF ————————————— data ————————————— EF   < ETL > EF ED]←Final reel
```

Format of RCA STANDARD Labels

1. BEGINNING TAPE LABEL (29 Characters)

The Standard Beginning Tape Label is twenty-nine characters in size and will always appear in message format, as follows:

$$< \overbrace{\bullet\ \text{IDENTIFICATION}}^{\text{9 ch's.}} \overbrace{\bullet\ \text{REEL-NUMBER}}^{\text{4 ch's.}} \overbrace{\bullet\ \text{DATE-WRITTEN}}^{\text{7 ch's.}} \overbrace{\bullet\ \text{PURGE-DATE}}^{\text{7 ch's.}} >$$

LABEL ITEMS

Name	Characters	Format
IDENTIFICATION (or ID)	9	•XXXXXXXX
REEL-NUMBER	4	•XXX
DATE-WRITTEN	7	•XXXXXX (MODAYR)
PURGE-DATE	7	•XXXXXX (MODAYR)

The Narrator obtains information for these items in the following manner:

a) ID - The 8-character literal appearing in the "VALUE" clause is used as the ID item.

b) REEL-NUMBER - This is initially set to 001, and incremented by "1" as each new reel is processed.

c) DATE-WRITTEN - today's date (supplied at program running time by user) is inserted as the date-written item.

d) PURGE-DATE - the number of days specified in the "ACTIVE-TIME" entry is added to todays date to arrive at the purge-date. (Note that ACTIVE-TIME must appear in the File Description for all output files using PURGE-DATE.)

2. ENDING TAPE LABEL (10 characters)

The Standard Ending Tape Label is ten characters in size and always appears in message format, as shown below:

$$< \overbrace{\bullet\text{BLOCK-COUNT}}^{\text{8 ch's.}} >$$

The BLOCK-COUNT item contains the number of "physical" blocks (or records) that appear on the reel. During the running program a count is kept of the number of "reads" from input files and "writes" to output files. When the end label is checked or created, the contents of these counters are taken as the "block-count." (Labels and control symbols are not included in the count.)

If records are "grouped" or "batched," each batch will be counted as one block in the file.

The format for this item is illustrated below:

Name	Characters	Format
BLOCK-COUNT	8	•XXXXXXX

Standard Label-Checking.

For Input files the Beginning-Tape-Label on each reel is automatically checked for Identification and reel-number. If these items are not correct, the program will stop and the user will be advised, via the monitor printer, to take corrective action.

For output files, a special check is made on all reels that are mounted on output tape stations to ascertain if these reels are "releasable," i.e., may be written upon. This is accomplished by reading the first information on each reel, which is assumed to be a Beginning-Tape-Label with a PURGE-DATE entry. If the Purge-Date in the label is not less than today's date, the program will stop with a print-out to the effect that this reel of tape is "not releasable."

Notes:

1. If the reel mounted on the Output trunk does not contain a label (for example, a work tape has been mounted), an EF symbol must appear as the first information on that tape. This will cause the program to bypass the Purge-date check.
2. The Purge-Date check will be performed on all reels that are used for output files, even though a particular output file was specified as having labels "omitted."

3. LABEL DESIGNED BY USER

If the Narrator user defines his own tape labels, he must adhere to the following rules:

A. Each File Description entry must contain one of the following clauses:

1. LABEL RECORDS ARE BEGINNING-TAPE-LABEL
2. LABEL RECORDS ARE BEGINNING-TAPE-LABEL, ENDING-TAPE-LABEL
3. LABEL RECORDS ARE OMITTED

B. A RECORD DESCRIPTION that specifies the format for the user's tape label(s) must appear within the FIRST File described in the DATA DIVISION. (These descriptions may not appear in any other File.)

The Beginning-Tape-Label must be defined as the FIRST Record Description.

The Ending-Tape-Label (if used) must be defined as the SECOND Record Description.

C. Label Items

The user may design his labels in any format, provided:

a) the labels are in message format

b) the Beginning-Tape-Label includes an IDENTIFICATION (or ID) item. The ID may not exceed twelve (12) characters, including an ISS if present.

If desired, the user's label may also include any or all of the remaining STANDARD label entries, provided these items are correctly spelled and are identical in format to that described on page 127. The advantage of using the standard items REEL-NUMBER, DATE-WRITTEN, PURGE-DATE and BLOCK-COUNT, is that these items will be automatically checked and created by the Input/Output Control as described under Standard labels.

For illustrative purposes, assume that the user's tape labels appear as follows:

BEGINNING LABEL:

$$< \overbrace{\bullet\ \text{ID}}^{\text{3 ch's.}}\ \overbrace{\text{SECURITY}}^{\text{2 ch's.}}\ \overbrace{\bullet\ \text{REEL-NUMBER}}^{\text{4 ch's.}}\ \overbrace{\text{FIRST-MSG-CRITERIA}}^{\text{8 ch's.}}\ \overbrace{\bullet\ \text{DATE-WRITTEN}}^{\text{7 ch's.}} >$$

ENDING LABEL:

$$< \overbrace{\text{LAST-MSG-CRITERIA}}^{\text{8 ch's.}}\ \overbrace{\bullet\ \text{BLOCK-COUNT}}^{\text{8 ch's.}} >$$

In this case, the first file in the Data Division would have as its file label entry: LABEL-RECORDS ARE BEGINNING-TAPE-LABEL, ENDING-TAPE-LABEL. In addition, the first two Record Descriptions in that file would appear as follows:

```
1 BEGINNING-TAPE-LABEL; SIZE 24; CLASS AN.
  2 ID; SIZE 3; CLASS ALPHABETIC.
  2 SECURITY; SIZE 2; CLASS NUMERIC; PICTURE J.
  2 REEL-NUMBER; SIZE 4.
  2 FIRST-MSG-CRITERIA; SIZE 8; PICTURE J.
  2 DATE-WRITTEN; SIZE 7.

1 ENDING-TAPE-LABEL; SIZE 16; CLASS NUMERIC.
  2 LAST-MSG-CRITERIA; SIZE 8; PICTURE J.
  2 BLOCK-COUNT; SIZE 8.
```

In the above example, the ID, REEL-NUMBER, DATE-WRITTEN and BLOCK-COUNT items would automatically be processed by the Input/Output Control. The processing of the other label items must be accomplished by USE subroutines.

For example, during the Beginning-Tape-Label processing for an output file, data for the FIRST-MSG-CRITERIA would be placed in the label area by the following USE subroutine statement:

```
MOVE EMPLOYEE-NUMBER TO FIRST-MSG-CRITERIA.
```

The format for files containing user-designed tape labels is shown below:

1. Beginning-Tape-Label only

SINGLE-REEL FILES

```
[< BTL >  EF ——————————— data ——————————— EF  EF  ED]
```

MULTIPLE-REEL FILES

```
[< BTL >  EF ——————————— data ——————————— EF  ED]<——Intermediate
                                                        reels

[< BTL >  EF ——————————— data ——————————— EF  EF  ED]<—— Final
                                                          reel
```

2. Beginning-Tape-Label and Ending-Tape-Label

Same as STANDARD label format (see page 127).

IMPORTANT NOTES

1. All files must follow the same tape label format; one file, for instance, cannot have standard labels while another has special labels designed by the user. For this reason, the Narrator assumes that the tape label format specified in the first file description entry applies to all files. Thus, if this entry states that labels are OMITTED, it will be assumed that labels are omitted for all files. (If label checking has been specified for the first file, subsequent files may contain the OMITTED clause.)

2. The label definitions in the first file are not considered as data records and should not, therefore, appear as record names in the "DATA-RECORDS ARE" clause.

PUNCHING REQUIREMENTS

501 COBOL NARRATOR PROGRAMS

Narrator programs are converted to punched paper tape for initial compilation. After initial compilation, the source program will be available on magnetic tape, and subsequent recompilations will use this magnetic tape as the basic source program. Modifications or corrections to the source program will, of course, be in paper tape format.

Each Narrator Division is punched in message format, as follows:

IDENTIFICATION DIVISION

```
< IDENTIFICATION DIVISION.>
< PROGRAM-ID . . . .       .>
< AUTHOR. . . . . .        .>
< INSTALLATION. . . .      .>
< DATE-WRITTEN . . . .     .>
< DATE-COMPILED . . .      .>
< SECURITY . . . . .       .>
< REMARKS . . . . .        .>

     EF (C/5)
```

ENVIRONMENT DIVISION

```
< ENVIRONMENT DIVISION.>

< CONFIGURATION SECTION.>
< OBJECT-COMPUTER.  5Ø1 MEMORY ADDRESS . . .    .>
< SPECIAL-NAMES.  BREAKPOINT-1 . . . , BREAKPOINT-2 . . .    .>

< INPUT-OUTPUT SECTION.>
< FILE-CONTROL.>
   < SELECT MASTER . . . . . .         .>
   < SELECT NEW-MASTER . . . .         .>
   < SELECT ERROR . . . . . . .        .>
< I-O-CONTROL.>
< RERUN . . . . . . . .        .>
< SAME AREA . . . . . .        .>
< SAME AREA . . . . . .        .>

     EF (C/5)
```

<u>DATA DIVISION</u>

```
< DATA DIVISION.>
< FILE SECTION.>

  < FD MASTER-FILE . . . . . . . . . . . . . . . . . .
     . . . . . . . . . . . . . . . DATA RECORDS ARE MASTER.>
    < 1 . . . . . . . . . . . .    .>
      < 2 . . . . . . . . .    .>
      < 2 . . . . . . . . .    .>
        < 3 . . . . .    .>
        < 3 . . . . .    .>
        < 3 . . . . .    .>
      < 2 . . . . . . . . .    .>
      < 88 . . . . . . . .    .>
      < 88 . . . . . . . .    .>

  < FD TRANSACTIONS . . . . . . . . . . . . . . . .
     . . . . . . . . . . . . . . . DATA RECORDS ARE CHECK, DEPOSIT.>
    < 1 CHECK . . . . . . . . .    .>
      < 2 . . . . . . . . .    .>
      < 2 . . . . . . . . .    .>
      < 2 . . . . . . . . .    .>
    < 1 DEPOSIT . . . . . . . .    .>
      < 2 . . . . . . . . .    .>
      < 2 . . . . . . . . .    .>
        < 3 . . . . . . .    .>
        < 3 . . . . . . .    .>
      < 2 . . . . . . . . .    .>
      < 2 . . . . . . . . .    .>

< WORKING-STORAGE SECTION.>
  < 77 . . . . . . . . . . . .    .>
  < 77 . . . . . . . . . . . .    .>
  < 1 . . . . . . . . . . . . .    .>
    < 2 . . . . . . . . . .    .>
    < 2 . . . . . . . . . .    .>
  < 1 . . . . . . . . . . . . .    .>
    < 2 . . . . . . . . . .    .>
      < 3 . . . . . . .    .>
      < 3 . . . . . . .    .>
    < 2 . . . . . . . . . .    .>
    < 88 . . . . . . . . .    .>
    < 88 . . . . . . . . .    .>

< CONSTANT SECTION.>
  < 77 . . . . . . . . . . . .    .>
  < 77 . . . . . . . . . . . .    .>
  < 77 . . . . . . . . . . . .    .>
  < 77 . . . . . . . . . . . .    .>

         EF (C/5)
```

PROCEDURE DIVISION

```
< PROCEDURE DIVISION.>
  < 1. SECTION.>
  < 2. MOVE A TO B.>
  < 3. MOVE C TO D.>
  < 4. IF . . .   .>
  < 5. READ . .   .>
  < 6. ADD . . .  .>

     EF (C/5)

     ED (A/8)
```

NOTE: If OWN CODE appears, an EF follows the last Procedural Statement, and an ED terminates the Own Code section. Punching requirements for pseudo-code instructions are described in the 501 Automatic Assembly System manual.

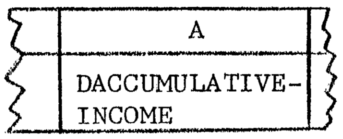
IMPORTANT NOTES

1. The maximum size (including SM and EM) of a Narrator message is 400 characters.

2. Particular attention should be paid to the use of hyphens and spaces when the program is punched. Section names, headers and KEY words, for example, must include hyphens where specified, i.e., FILE-CONTROL, INPUT-OUTPUT SECTION, ACTIVE-TIME, etc.

   At least one space must separate words and periods are critical punctuation characters.

3. If data-names or literals have been "squeezed" into columns on Assembly Pseudo-code Program Sheets, the key-punch operator must be instructed to punch this information within the boundaries of that column. For example:



A
DACCUMULATIVE- INCOME

must be key-punched as shown below:

•DACCUMULATIVE-INCOME

APPENDIX A

RESTRICTED & RESERVED WORDS

The following words cannot be used by the programmer when assigning data-names:

ADVANCING	LOCATION	TALLY
ALL	MODE	THAN
ARE	OF	THEN
AT	ON	TIMES
ASSEMBLER	PLACES	WHEN
CHARACTERS	PROCEDURE	WITH
CONTAINS	PROTECT	ZERO
DOLLAR	SPACE	ZEROES
IS	SPACES	ZEROS
LINES	STANDARD	

In addition, it is also recommended that the user refrain from using KEY or Optional words as data-names, because of the possibility of creating ambiguities within the English statements.

The following words have special significance, and may only be used as specified in this manual:

Tape Labeling	Procedure Division	Data Division
ID	ALL	FILLER
IDENTIFICATION	TALLY	
REEL-NUMBER	ZERO	
DATE-WRITTEN	ZEROS	
PURGE-DATE	ZEROES	
BLOCK-COUNT	SPACE	
	SPACES	

APPENDIX A

LIST OF KEY AND OPTIONAL WORDS

A. IDENTIFICATION DIVISION

Key Words

AUTHOR	INSTALLATION	REMARKS
DATE-COMPILED	PROGRAM-ID	SECURITY
DATE-WRITTEN		

B. ENVIRONMENT DIVISION

Key Words

ADDRESS	OPTIONAL	SELECT
ASSIGN	PRINTER	SPECIAL-NAMES
IS	RECORDS	TAPE
MEMORY	REEL	TAPES
MULTIPLE	RERUN	TCP
NO	RESERVE	THROUGH
OBJECT-COMPUTER	SAME	THRU
ON		

Optional Words

ALTERNATE	END	FOR
AREA	EVERY	STATUS
AREAS	OF	

C. DATA DIVISION

Key Words

ACTIVE-TIME	FD	RECORDS
ALPHABETIC	FLOAT	RECORDING
ALPHANUMERIC	JUSTIFIED	REDEFINES
AN	LABEL	RIGHT
BEGINNING-TAPE-LABEL	LEAVING	SIGN
BLANK	NUMERIC	SIGNED
BLOCK	OCCURS	SIZE
CHECK	OMITTED	STANDARD
CLASS	PICTURE	SUPPRESS
COPY	POINT	TO
ENDING-TAPE-LABEL	RECORD	VALUE
ID		ZERO
IDENTIFICATION		

Optional Words

ARE	IS	PLACES
CHARACTERS	OF	PROTECT
CONTAINS	LOCATION	TIMES
DOLLAR	MODE	WHEN

APPENDIX A

D. PROCEDURE DIVISION

Key Words

ACCEPT	FIRST	POSITIVE
ADD	FROM	PROCEED
AFTER	GIVING	READ
ALL	GO	REEL
ALTER	GREATER	REPLACING
AND	IF	REWIND
BEFORE	INPUT	ROUNDED
BEGINNING	INTO	RUN
BY	LABEL	SECTION
CLOSE	LEADING	SIZE
DEPENDING	LESS	STOP
DISPLAY	LOOP	SUBTRACT
DIVIDE	MONITOR	TALLYING
ELSE	MOVE	THROUGH
END	MULTIPLY	THRU
ENDING	NEGATIVE	TIMES
ENTER	NO	TO
EQUAL	NOT	TRUNK
EQUALS	NOTE	UNEQUAL
ERROR	NUMERIC	UNTIL
EXACTLY	OPEN	UPON
EXAMINE	OTHERWISE	USE
EXCEEDS	OUTPUT	WITHIN
EXIT	PAGE	WRITE
FILLING	PAPER-TAPE-READER	ZERO
FIND	PERFORM	

Optional Words

ADVANCING	LINES	WITH
ASSEMBLER	ON	STANDARD
AT	PROCEDURE	THAN
IS	RECORD	THEN