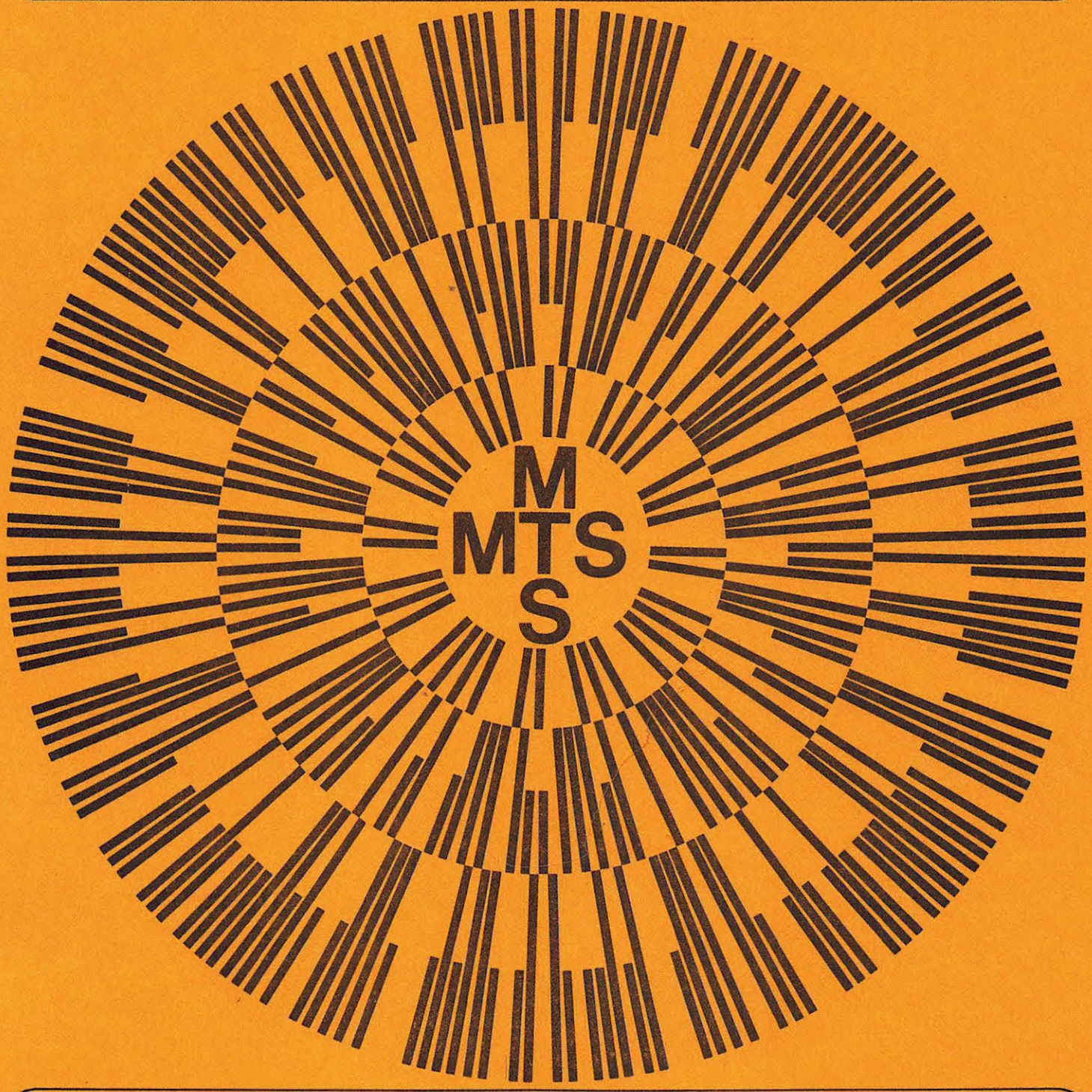
THE UNIVERSITY OF ALBERTA
COMPUTING CENTER
PUBLICATION



FORTRAN (G AND H)

ACKNOWLEDGEMENTS

THIS MANUAL WAS LARGELY COMPILED FROM MATERIAL PREPARED BY THE STAFF OF THE UNIVERSITY OF MICHIGAN COMPUTING CENTER. THEIR DOCUMENTATION WAS INVALUABLE AND WE ARE INDEBTED TO THEM FOR ALLOWING US TO USE IT. IN PARTICULAR, THE FOLLOWING WERE MOST USEFUL:

MTS USERS' MANUAL, SECOND EDITION, VOLUMES I AND II

MTS USERS' MANUAL, THIRD EDITION, VOLUME 2

INTRODUCTION TO MTS AND THE COMPUTING CENTER (FLANIGAN)

COMPUTING CENTER NEWS ITEMS

COMPUTING CENTER MEMOS

THE COMPUTING CENTER WISHES TO PERSONALLY ACKNOWLEDGE THE ASSISTANCE OF MIKE ALEXANDER AND DON BOETTNER WHO HELPED US TO ESTABLISH MTS AT THE UNIVERSITY OF ALBERTA.

ACKNOWLEDGEMENT SHOULD ALSO BE MADE TO THE COMPUTING CENTRE, UNIVERSITY OF BRITISH COLUMBIA, FOR INFORMATION OBTAINED FROM SOME OF THEIR DOCUMENTATION AND TO I.B.M., WHOSE MANUALS PROVIDED CERTAIN SECTIONS FOR OUR MANUALS.

FORTRAN (G AND H)

MAY, 1970

DISCLAIMER

This MTS manual is a combination of earlier manuals, update notices, memos and limited experience with the system itself. Because of this, certain discrepancies are bound to occur and the Computing Center would appreciate being notified of all differences between what this manual says and what the system actually does.

This publication is intended to represent the current state-of-the-system. However, it should not be construed as an obligation to maintain the system as so stated. The MTS system, like most good systems, is continually being improved. As a result, additions, extentions, changes and deletions will occur. Notice of such changes will be made and provision for a manual updating service has been planned.

Errors, comments and suggestions should be sent to:

Information Coordinator
Computing Center
University of Alberta

FORTRAN (G AND H)
MAY, 1970

FORTRAN (G AND H)

TABLE OF CONTENTS

SUMMARY

*FORTG

Logical I/O Units

SCARDS = Source Input
SPRINT = Compiler Listing and Diagnostics
SPUNCH = Object Modules
SERCOM = Error Summary Output

The defaults for the logical I/O units SCARDS, SPRINT, SPUNCH and SERCOM are *SOURCE*, *SINK*, -LOAD# and *SINK*.

Compiler Options

PAR=[BCD / **EBCDIC**] [,DECK (D) / **NODECK** (NOD)] [,LIST (L) / **NOLIST** (NOL)] [,MAP (M) / **NOMAP** (NOM)]

[,SOURCE (S) / **NOSOURCE** (NOS)] [,ERR / **NOERR**] [,SML / **NOSML**] [,NAME={aaaaaaaa / **MAIN**}]

[,LINE={nn / **57**}] [,ORL={S / L}] [,EXEC = {0 / 4 / 8}] [,LOAD / **NOLOAD**]

In the above options, the abbreviation, if any, is enclosed in parentheses while the default option is underlined. The standard default options for batch users are:

PAR = S,NOM,NOL,LOAD,NOD,EBCDIC,NAME=MAIN,LINE=57

and for conversational users are:

PAR=NOSML,ERR,LOAD,NOD,EBCDIC,NAME=MAIN,LINE=57.

*FORTH

Logical I/O Units

SCARDS = Source Input
SPRINT = Compiler Listings and Diagnostics
SPUNCH = Object module if DECK specified
0 = Object module if LOAD specified.

The defaults for the logical I/O units SCARDS, SPRINT and SPUNCH are *SOURCE*, *SINK* and -LOAD#. If the LOAD option is specified, then the logical I/O unit 0 must be assigned.

Compiler Options

PAR = [BCD (B) / EBCDIC (EB)] [,DECK (D) / NODECK (NOD)] [,EDIT (E) / NOEDIT (NOE)]

[,LIST (L) / NOLIST (NOL)] [,LOAD (LO) / NOLOAD (NOLO)] [,MAP (M) / NOMAP (NOM)]

[,XL / NOXL] [,LINECNT={nn / 58}] [,NAME=aaaaaaaa / MAIN] [,OPT={0 / 1 / 2}]

[,ID (I) / NOID (NOI)] [,SOURCE (S) / NOSOURCE (NOS)] [,XREF (X) / NOXREF (NOX)]

The abbreviations, if any, for the options are enclosed in parentheses while the default option is underlined.

LANGUAGE DESCRIPTION

References

1) IBM SYSTEM/360 FORTRAN IV LANGUAGE, FORM C28-6515.
2) IBM SYSTEM/360 FORTRAN IV (G & H) PROGRAMMER'S GUIDE, FORM C28-6817.
3) IBM SYSTEM/360 FORTRAN IV (G) PROGRAMMER'S GUIDE, FORM C28-6639.
4) IBM SYSTEM/360 FORTRAN IV (H) PROGRAMMER'S GUIDE, FORM C28-6602.

MTS Conventions and Restrictions

1. Data Set Reference Numbers

The legal data set reference numbers (DSRN'S) in MTS FORTRAN are 0 through 9 and correspond to the appropriate MTS logical I/O units. The READ, PRINT and PUNCH statements described in Appendix B of the IBM System/360 FORTRAN IV Language SRL (C28-6515) are connected to the logical devices, SCARDS, SPRINT and SPUNCH, respectively. In IBM systems these statements usually default to data set reference numbers 5, 6 and 7, respectively. In addition, the logical device SERCOM is used for error comments, and the PAUSE and STOP statements. All appropriate logical devices should be assigned when the $RUN command is issued for the FORTRAN object module. At open time, any legal data set reference number which has not been assigned will be defaulted to SCARDS for input and SPRINT for output. For output operations, illegal data set reference numbers act like *DUMMY*, i.e., they automatically serve as infinite waste baskets. For input operations illegal data set reference numbers result in the comment

```
READ DSRN XXXXXXXX ?
```

followed by a return to command mode. By issuing a $RESTART command the program will resume execution by ignoring the READ statement.

2. The Direct Access Statements

The FIND and indexed READ/WRITE statements may be applied to any logical I/O unit, but will be ignored (by the device support routines) unless it has been assigned to a file. The integer index is interpreted as an MTS line number times 1000, e.g., to begin at line 5.7 the index should be 5700. On any index I/O operation, only the first data set record will be read or written indexed; any subsequent data set records will be obtained sequentially. For example, the following statements would read lines 10 and 11, if the default increment of 1 is used.

```
      READ(5'10000,100)A,I
100 FORMAT (G15.8/I10)
```

The following statements would erase lines 1 through 6 from a file assigned to 5

```
      WRITE (5'1000,100)
100 FORMAT (/////)
```

Since six zero length lines will be written. This example also presumes the default increment. The FIND statement can be used at any time and will cause the next sequential READ/WRITE statement to be done indexed. If the next I/O statement after the FIND is indexed, the index of the I/O statement will override that given in the FIND.

If an index given in a READ statement, or if an outstanding FIND statement causes a sequential READ to be done indexed, and the index represents a non-existent line then an end-of-file situation is recognized. This will either result in successful termination of the program or will exit via the END= parameter supplied in the READ statement. Note that the END= parameter cannot be given in an indexed READ statement. Accordingly, the sequence FIND, sequential READ is recommended.

The DEFINE FILE statement is not meaningful in MTS and though it will continue to compile, execution of such a statement will solicit an appropriate comment (IHC214).

3. The REWIND Statement

The FORTRAN REWIND statement may be applied to a data set reference number assigned to a sequential file with the desired result. A reference to a DSRN in a REWIND statement causes the associated data set to be opened.

4. The BACKSPACE and ENDFILE Statements

The BACKSPACE and ENDFILE statements will be ignored unless the logical I/O unit has been assigned to a magnetic tape and the data set has been opened, e.g., a READ or WRITE statement has already referenced the data set. Further, the BACKSPACE will also be ignored if the tape is currently positioned immediately after an end-of-file. This feature is for compatibility with IBM systems and stems from their treatment of different files on the same tape as different data sets.

5. The STOP Statement

The STOP statement has been changed so that after printing the stop message on SERCOM it will terminate execution in such a manner that a storage dump will not be given and execution cannot be resumed.

6. The PAUSE Statement

The PAUSE statement will no longer write the pause message on the operators console. In batch mode, PAUSE will print the pause message on SERCOM and continue program execution, i.e., it can be used to print comments only. In conversational mode, after printing the pause message on SERCOM, program execution will be suspended in such a manner that it may be resumed using the RESTART command.

7. The ERR= Parameter

The ERR= parameter exit from both sequential and indexed READ statements will never be taken. In IBM systems, this parameter implements SYNAD exits from the data management routines. Similar situations in MTS normally result in program termination.

8. Data Set Record Lengths

The data set record length has been expanded to separate input and output record lengths. The role of these parameters is described in the section devoted to FORTRAN and data set records.

9. Unformatted Record Format

The format and amount of control information placed in each data set record generated by an unformatted WRITE statement has been changed to conform to OS unblocked variable length record format. A detailed description is given in the section on FORTRAN and data set records. By default, unformatted READ operations presume the same record format. It is possible to read records written according to the old greenword format by calling the subroutine SETGRE(DSRN), which takes as its single argument a fullword integer DSRN.

10. Tape Support

The FORTRAN I/O library will now support any size tape record legitimate to the tape support routines. When opening any data set, GDINFO is used to determine the device type of the data set. If the data set is a tape, the tape support routines will be requested to pass back the record size specified in the $RUN *MOUNT command used to obtain the tape. This number is used to set the input and output record lengths and if it exceeds 256 a buffer private for the data set will be allocated.

11. Implicit and Explicit Concatenation

The first reference to a data set causes it to be opened, a procedure which is dependent on the file/device type. Since this dependence is maintained through such things as the record lengths and the admissibility of the positioning statements; and since the library has no knowledge of when and if concatenation takes place, care should be exercised when concatenating unlike data sets. It should also be noted that when DSRNs default to SCARDS, that they are SCARDS with respect to concatenation. Hence, a $CONTINUE WITH in SCARDS will change all defaulted DSRNs.

12. Program Interrupts

Program interrupts will no longer be fielded by the FORTRAN library. This decision has at least two consequences that represent incompatibilities with the old library. First, the subroutine OVERFL and DVCHK cannot properly function and have accordingly been discontinued. Second, since MTS will take the program interrupts, strings of floating-point exceptions are not possible since MTS will suspend execution via ERROR for any and all program interrupts.

13. Formatted Fields

When reading, any field to be converted according to a format code other than A is scanned for commas and semi-colons, both of which are normally illegal in these fields. If neither of these "separators" is found, the conversion will be performed just as it has been in the past, with blanks being ignored or treated as zeros according to the specifications in the last call to FCVTHB. If a comma is found, the field width is changed to terminate with the last character before the comma. In the case of floating-point fields, the number of places to the right of the decimal point is also set to zero. If the resulting field width is zero, the list item is left unchanged. If a semi-colon is found, processing proceeds as if a comma had been found; however, after the current list item has been taken care of, I/O processing of the current READ statement will be suspended. Accordingly, any remaining list items will remain unchanged. In the same context, if a field extends beyond the record read and characters remain, then the format code will be modified to just complete the field within the current data set record. Note that the last number in a string will usually be followed by some trailing blanks which will be treated as zeros be default. Hence, the last field of a string should be terminated by either a comma or semi-colon.

To illustrate these features, consider the following program:

```
10   READ 100,A,N,M
100  FORMAT (F20.8,I5,I3)
     PRINT 200,A,N,M
200  FORMAT (F10.5,2I6)
     GO TO 10
     END
```

In the following, the odd numbered lines are input and the even numbered lines are the resulting output.

```
1.,5,6;
  1.00000      5     6
1.5;
  1.50000      5     6
,111113,
  1.50000 11111      3
,,2
  1.50000 11111    200
```

The first example is evident. The second example illustrates premature termination of I/O processing that can be accomplished using a semi-colon. In the third example, the first operand is omitted and remains unchanged, the second operand fills the entire field and consequently, the field is not terminated by a comma. Note, if the line has been given as ",11111,3," then the third list item would not have been changed, since the additional comma would not have been found until the I3 field was scanned. The final example illustrates why a terminal comma or semi-colon is advisable, since the intent was that M=2, not 200.

14. Pseudo Sense Lights

The library subroutine SLITE and SLITET which implement the pseudo sense lights which carried over from the IBM 7000-series FORTRAN have been dropped.

15. Set Data Set Record Lengths

The subroutine SETDSR (DSRN,IRL,ORL) may be used to alter the input and output record lengths. All three agruments are fullword integers.

16. Set Stare Count

SETSTA (DSRN,TIMES) can be used to stare at the next input line. The stare facility is only available for lines read under format control and is reset by any other operation on the data set. The two arguments are both fullword integers, but the stare count is taken modulo 256. If the stare count is set and an end-of-file is encountered then the end-of-file will be repeated. Calling SETSTA will automatically allocate a private I/O buffer for the data set. Calling SETSTA with a zero count will reset an outstanding stare count so that the next read operation will look at the next data set record.

17. BLOCK DATA Subroutines and COMMON BLOCKS

BLOCK DATA subroutines must precede other routines (including the main program) in the FORTRAN source deck in order that data be correctly placed in the various COMMON blocks used. Execution will start at the routine immediately following the BLOCK DATA subroutine so the main program should come there.

The reason for this restriction is that the MTS one pass loader only uses the first appearance of COMMON blocks in setting up the blocks. Any subsequent appearance of the COMMON blocks are essentially ignored.

Another restriction introduced for the same reason is that the first definition of a COMMON block in a FORTRAN source deck must be the longest definition of the block.

18. FORTRAN and Data Set Records

The vast majority of FORTRAN programs may be written with little or no knowledge of how FORTRAN I/O operates beyond the fact that printer lines cannot exceed 132 characters, and even this is superfluous if NAMELIST I/O is employed. This simplicity of operation is due to the rigidity of the I/O devices being used, i.e., card readers and printers. These devices cannot be rewound or backspaced and allow a very simple and limited interpretation of the data set records. This simplicity disappears as soon as the I/O devices being used become more complex, e.g., terminals, disks, magnetic tapes. The inclusion of terminals in this category is due primarily to the fact that when reading from a terminal, only those characters typed are transmitted; the blanks which the typist may or may not want appended to his input line are not transmitted by the terminal as they would be from a card reader. The effects of this situation would become immediately apparent if the FORTRAN I/O routines did not pad terminal input lines with blanks to form an 80 character line image. This has the effect of making terminal I/O and normal batch I/O appear the same to the user.

With tapes and disks the problems are compounded, as these devices are capable of not only input and output but also allow positioning, e.g., rewinding. Currently, MTS imposes a limit of 255 characters per line for disks and 32768 characters per record for tapes. The same question that causes confusion with terminal input arises with these devices: How many, if any, padding blanks should be added to input lines? To solve this and other problems which arise, it is the programmer's responsibility to ensure that the FORTRAN records generated by these formats and I/O lists are meaningful in the context of the data set records which are transmitted to and received from the various I/O devices. The remainder of this section, therefore, gives a description of FORTRAN and data set records and how they are related.

A FORTRAN format and the associated I/O list define a FORTRAN record(s) as follows:

(1) If the format contains no slashes or additional parenthesis, the FORTRAN record is defined by the beginning of the format (left paren) to the end of format (right paren).

Example: The format (I10,3I20) defines FORTRAN records containing 70 characters. Note that a format defines a FORTRAN record, while the I/O lists used with the format determine how many FORTRAN records are read or written. With the above example format, a five element output list would cause two FORTRAN records to be written, with the second record consisting of a 10 character integer field. On the other hand, an input list containing 20 elements would cause 5 FORTRAN records to be read.

(2) If slashes appear within a format statement, then each of the following defines a FORTRAN record: beginning of the format to the first slash, from one slash to the next slash, and from the last slash to the end of the format.

Example: The format (10A8/10I3/5F15.5) defines three FORTRAN records: The first 80 characters long, the second 30 characters long and the third 75 characters long. Again, it is the I/O list that determines how many FORTRAN records will be read or written.

(3) If more than one parenthesis level appears within a format, the first FORTRAN record is defined by the beginning of the FORMAT to the end of the format. If the I/O list is not satisfied with the single FORTRAN record all subsequent FORTRAN records will be constructed based upon the format specifications beginning with the first level-0 left parenthesis (including any preceding repetition factor) that precedes the end of the format.

Example: The format (I10,2(I10,2(I5)),I10) would cause the first record to be operated on according to the format as given; however, all subsequent records would be processed according to a format of

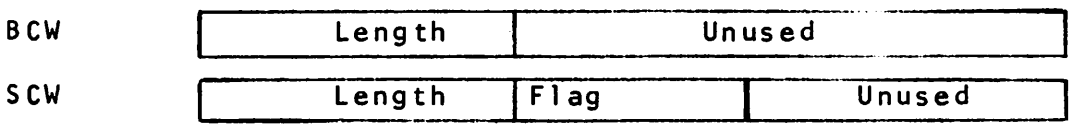
(2(I10,2(I5)),I10).

The format (I10,(F10.5),6(F10.5)) would cause all secondary records to be formatted according to 6(F10.5). Because of the limited parenthetical expressions allowed, these two examples illustrate both possibilities with regard to illustrating the definition of the first level-0 left parenthesis.

(4) A description of the FORTRAN records produced by formats containing both parenthetical expressions and slashes can be obtained from extensions of the above remarks if it is kept in mind that the slashes are not relevant when locating the first level-0 left parenthesis after reaching the end of the format. It should be kept in mind, however, that the terminal right parenthesis acts like a slash.

When operating under format control there is a one-to-one correspondence between FORTRAN records and data set records. When writing, the maximum size of the data set record, and hence also the FORTRAN record, is the output record length. When reading, the maximum size is the larger of the input and output record lengths. Any input record containing fewer characters than the number provided for by the input record length will be padded with blanks to that length. Records in excess of the input record length are not truncated.

An unformatted FORTRAN record is defined entirely by the I/O list, its length in bytes being the sum of the lengths of the I/O list items. Depending upon the length of this record, however, many data set records may be generated. Each of these data set records will be prefixed with two control words formatted as follows:



The BCW is the first word of each data set record and the BCW length equals the number of bytes in the data set record including both the BCW and SCW. The SCW is the second word, and the SCW length equals the number of bytes in the data set record including the SCW but excluding the BCW word. The flag byte in the SCW indicates whether the FORTRAN record spans into and/or out of the data set record, and is coded as follows:

01 FORTRAN record spans into next data set record.
02 FORTRAN record spans from last data set record.

As an example, if the length of the data set record is large enough to contain the entire FORTRAN record then the flag byte would be 00. Otherwise, the FORTRAN record will be broken into a series of data set records, the first with a flag byte of 01, the last with a flag byte of 02 and the intermediate records having a flag byte of 03. The maximum size of an output record is the larger of the input and output record lengths, which includes the eight bytes of control information. When reading, the BCW length is ignored and the SCW length is compared with the length returned by the MTS I/O routines. If the record is too short it is padded with blanks to the number of bytes given in the SCW word.

In the following chart of the default data set record lengths, SIZE denotes the integer given in the parameter field of the $RUN *MOUNT command:

DEVICE TYPE	INPUT RECORD LENGTH	OUTPUT RECORD LENGTH
TAPE	SIZE	SIZE
FILE	255	255
OTHER	largest existing record in the file or 255 if record length is 0.	

The subroutine SETDSR is available to alter the default data set record lengths. The arguments are three full word integers and are in order: the data set reference number, the input record length, and the output record length. At open time, a buffer large enough for a maximum sized record is allocated so that changing the record lengths should not necessitate buffer allocation. The values given to SETDSR are not however, checked to ensure that they are less than the legal maximum. As an example, to set data set six (6) for 71 character lines one should code the following FORTRAN statement:

```
CALL SETDSR (6,71,71).
```

In this context, it should be noted that all buffers are at least 256 characters long. Implicit or explicit concatenation of unlike data sets can cause trouble since the FORTRAN I/O routines are not notified when concatenation takes place. This sort of thing can cause problems not only in the FORTRAN I/O positioning statements, but also in normal READ/WRITE statements due to differing input/output record lengths.

19. FORTRAN Library Changes

The module DEBUG# has been corrected, so that it is now possible to invoke the INIT option for all variables, even those passed as arguments to external functions.

The FORTRAN I/O library will now recognize the device type corresponding to a sequential file. This change will allow the REWIND statement to be applied to a sequential file and alters the input and output record lengths currently defaulted for sequential files.

The default input and output record lengths for seqential files will be equal to the number of characters (bytes) in the largest existing record in the file, or if the file is empty 255. Up until now the input record length has been 80 and the output record length 132. As with all other device types the routine SETDSR may not be used to increase the input and/or output record lengths beyond the maximun of 255 and the default values.

The following example program illustrates one method of conditioning the FORTRAN I/O routines to write large records into a sequential file.

```
      REAL    A(30,30),X(30),LAMDA
      INTEGER*2  K4TH/4000/
10    CALL WRITE  (A,K4TH,0,1,5)
20    REWIND 5
        ...
30    WRITE(5) LAMDA,X,A
        ...
      END
```

When statement 10 is executed the FORTRAN I/O library has not as yet had occasion to deal with logical I/O unit 5 and hence has not generated the input record length. This call to the WRITE subroutine will write a single 4000 byte record (presumably 5 is assigned to a sequential file) into the file. Though the array A does not contain 4000 bytes, because of the standard FORTRAN storage allocation technique, starting at A there will most likely be 4000 addressable bytes for the WRITE subroutine. Note that this write will append this 4000 byte record to the end of the sequential file. The execution of the REWIND will cause the I/O library to open the data set assigned to logical I/O unit 5 at which time it will discover that it is a sequential file containing at least one 4000 byte record. Accordingly, the input and output record lengths will be set to 4000 and an appropriate buffer will be allocated. The write statement labelled 30 will write a single 3732 byte record, 8 bytes of control information, 4 bytes for LAMDA, 120 bytes for the vector X and 3600 bytes for the matrix A. If statement 10 has been omitted and the sequential file was empty, this same write statement

would write a total of 16 records, each containing 8 bytes of control information and at most 244 bytes of data, i.e., at most 61 numbers (words) per record. Also note that if a program is now written to read this file that the input and output record lengths will default to 3732, not 4000.

20. Error Comments that FIOCS will produce

```
FIOCS   READ DSRN  XXXXXXXX?
FIOCS - BUFFER OVERFLOW ON READ
FIOCS - NO BUFFER SPACE AVAILABLE
FIOCS - OUTPUT DEVICE IS FULL
FIOCS - END OF FILE ON N
```

PROCESSOR DESCRIPTIONS

*FORTG

Description: The IBM FORTRAN G compiler with a modified system interface module to allow it to operate in MTS.

Purpose: To compile legitimate FORTRAN IV G source programs and perform auxiliary services.

Logical I/O Units Referenced:

SCARDS - Input data set consisting of parameter cards, source modules, and/or object modules.
SPRINT - Compiler listings and diagnostics. The amount and type of information is controlled by the parameters SOURCE, MAP, LIST, and ERR.
SPUNCH - Object modules produced by the compiler. This output is controlled by the parameters LOAD and DECK and by the type of the file or device assigned to it.
SERCOM - Error summary output.

Compiler Options:

BCD - Source cards in BCD (026 keypunch).
DECK or D
- Produce sequence ID object DECKS on SPUNCH device.
EBCDIC - Source cards in extended BCD code (029 keypunch).
ERR - Produce diagnostic output even when NOSML.
EXEC=(0/4/8)
- Stop the job if the return code is greater than EXEC.
LINE=DD Set number of lines/page for SPRINT output.
LIST or L
- Produce object listing on SPRINT.
LOAD - Produce complete object file in SPUNCH file.
MAP or M
- Produce storage map on SPRINT.
NAME=AAAAAAAA
- Set name given to main programs.

ORL=S or L
- Set output record format for SPRINT

SOURCE or S
- Produce source listing on SPRINT.

SML - Equivalent to S,M, and L.

Any of the non-keyword parameters, i.e. those without equal signs, may be given the prefix NO with the obvious meaning. The EXEC parameter may be used to terminate a job due to unsuccessful compilation in batch mode. If the compiler return code indicates the most serious compilation error and a condition code greater than that assigned to the EXEC parameter then the task will be terminated immediately after the last source program is compiled. This parameter is ignored in conversational mode, where the return code may be obtained via a $DIS GR15 command if it is not otherwise apparent. If the EXEC parameter is not specified then task execution will proceed regardless of compilation errors.

If the BCD option is selected, statement numbers passed as arguments must be coded as $N and $ must not be used as an alphabetic character. Normally the EBCDIC option is used, the coding of statement number is &N, and $ is a legal alphabetic character. Most terminals and the 029 keypunches produce the EBCDIC code.

The standard default options for batch users are

SOURCE,NOMAP,NOLIST,LOAD,NODECK,EBCDIC,NAME=MAIN, LINE=57

while for conversational users they are

NOSML,ERR,LOAD,NODECK,EBCDIC,NAME=MAIN,LINE=57

The ORL parameter will be defaulted according to whether the user is in conversational or batch mode unless it is given in the PAR field of the $RUN command. Once set, this parameter cannot be changed.

The default options combined with the options given in the PAR=field of the $RUN *FORTG command form the set of options to which each parameter line in the input stream is applied. Consider the following batch examples:

```
$RUN *FORTG   PAR=M
     options:  S,M,NOL,LOAD,NOD
PAR=D
     options:  S,M,NOL,LOAD,D
PAR=
     options:  S,M,NOL,LOAD,NOD
```

Note that a parameter line with no text resets the options to those obtained from the PAR= field of the $RUN command.

SCARDS Input

The type of each line in the input stream is determined as follows:

(1) Parameter line - if the first four characters are "PAR=".
(2) Object line - if the first character is the non-printing HEX character 02 (card punch 12-9-2).
(3) MTS command - if the first character is a dollar sign ($).
(4) Source line - if neither of the above.

Regardless of its type, no line may exceed 80 characters. Only the first 72 characters of source and parameter lines are recognized as text, positions 73-80 being regarded as an ID field.

Parameter lines are used to alter the compiler options, e.g. DECK, LIST, and may appear anywhere. The text portion of a parameter line should not contain imbedded blanks. The options appearing in a parameter line represent temporary revisions to the default options which result from combining the options that appear in the PAR= field of the $RUN command and the standard default options. The revised options will take effect beginning with the next source module, and remain in effect until the first source module after the next parameter line.

Object lines are tolerated, not processed. If the load option is on, the object lines are transmitted unchanged to the SPUNCH file; otherwise, they are simply ignored. The occurrence of an object line always forces termination of the current source module.

If an input line contains a $ in column 1, it is presumed to be a command. If command lines precede the first source statement, they will be executed before any compilations. Any other

occurrence serves as an end-of-file and hence terminates the source module being compiled. In any case, the RESTART command must be issued to resume the compiler operation after each command execution. The primary purpose of this feature is to eliminate the necessity for the $ENDFILE card in batch operation. Note that when the EXEC parameter is used the QUIT routine is called as soon as any errors of the appropriate level are found, hence placing commands between programs may result in premature termination.

SPRINT Output

Output to this logical I/O unit is controlled by the parameters

SOURCE(S),MAP(M), LIST(L), SML, ERR, ORL

or their negatives which are obtained by adding the prefix NO. The parameter ORL has no negative and must be given as either ORL=L or ORL=S.

If the parameters are equivalent to NOSML, no output will be produced on SPRINT. In the past these parameters have produced the diagnostics and the TOTAL MEMORY ... line; to obtain this type of output now the parameter ERR must be given. The ERR parameter is ignored unless the equivalent of NOSML is explicitly given, in which case only compiler diagnostics will be printed. For conversational users, the default parameters are NOSML and ERR.

The output record length parameter, ORL, controls the length of the SPRINT output lines. In the past, only the 120 character long output format has been available; however, to adapt to 71 character devices like teletypes a short format is now also available. The permissible setting for ORL are L and S, in the PAR= field of the $RUN command: if not given, it will default according to whether the user is in conversational or batch mode. Once set, this parameter cannot be changed, i.e., attempts to do so will be ignored. In the short format, the SOURCE listing lines are squashed to 78 characters, the MAP listing lines are wrapped, and the OBJECT listing lines are squashed to 71 characters.

SERCOM output

For each program for which diagnostics are given, a line of the form

ERROR SUMMARY FOR PROGRAM *NAME* CNT0 CNT4 CNT8

is printed on SERCOM where *NAME* is replaced by the name of the program and CNTn is replaced by a four digit integer count of the errors of level n within that program. For conversational operation the line is of the form

NAME CNT0 CNT4 CNT8

SPUNCH Output

Output to this logical I/O unit is controlled by the parameters LOAD and DECK(D) and the type of file/device to which it is assigned. To facilitate selective updating of the object modules during compilation, they must be placed in a line file. If SPUNCH is assigned to a line file it can be used directly; however, if it is not, the compiler obtains and empties the temporary file -LOAD#. Hence there is always a SPUNCH line file regardless of the assignment of this logical I/O unit. The LOAD parameter controls output to the actual device assigned to SPUNCH.

If the LOAD option is on, all object modules produced by the compiler and those found in the input stream are placed in the SPUNCH file. If the LOAD option is not on, then (1) object modules in the input stream will not be transferred to the SPUNCH file, although they will be read; and (2) the SPUNCH file will be re-used for each object module produced by the compiler.

If the DECK option is specified then the compiler will place sequence ID in the SPUNCH file records. If SPUNCH is assigned to a file then the object modules produced by the compiler will be copied to the SPUNCH device after the completion of the appropriate source module compilation.

If it is intended to execute the resulting program, the LOAD option must be on and is hence a default option, If you understand the preceding, then, you will probably never assign SPUNCH to a file and specify NOLOAD and DECK.

ASSIGNMENT OF SPUNCH FOR BATCH USERS

In general, batch users need not assign SPUNCH. The system will default SPUNCH to *PUNCH* the card punch, and FORTRAN will accordingly open and empty the file -LOAD#. The complete object program will then be placed in the file -LOAD#, and each object module produced with the DECK option will be copied to the card punch. A subsequent $RUN -LOAD# command will then cause the program to be executed. Note that since parameter cards may be inserted in the input stream, it is now possible to obtain object decks on a selective basis.

If only object decks are desired, it suffices to specify NOLOAD and DECK on the $RUN *FORTG card and let SPUNCH default. The options NOLOAD and DECK will then be default options for all parameter lines in the input stream, and any object modules encountered in the input stream will be spaced over.

ASSIGNMENT OF SPUNCH FOR CONVERSATIONAL USERS

Usually, SPUNCH need not be assigned. Though the system will not default SPUNCH, when FORTRAN finds it unassigned it will open and empty the file -LOAD#. The complete object program will be placed in the file -LOAD# and a subsequent $RUN -LOAD# command will initiate program execution.

If it is desired to place the object modules in a permanent line file simply assign SPUNCH to said file. Under these circumstances, sequence ID may be obtained by also specifying the DECK option. The file -LOAD# will not be opened.

If SPUNCH is assigned to a sequential file or a device, then -LOAD# will be opened and emptied, the LOAD option will control output to the SPUNCH file, and the DECK option will perform the same service for the SPUNCH device.

TERMINAL ERROR CONDITIONS

If the initialization is unable to allocate the file -LOAD#, the compiler will abend. The compiler will abend if a request for a core storage block cannot be satisfied, or if the fixed tables in IEYROL are exceeded. Any program interrupt other than floating-point overflow or underflow causes compiler termination. The specific error comments are:

```
FORTRAN COULD NOT SUCCESSFULLY ALLOCATE
THE FILE -LOAD#?
FORTRAN COULD NOT OBTAIN SUFFICIENT CORE SPACE?
FIXED TABLES WITHIN FORTRAN EXCEEDED?
```

RETURN AND CONDITION CODES

When the compiler returns to MTS it leaves a return code in general register 15, which may be displayed by issuing the command $DIS GR15. The meaning of these return codes are as follows:

CODE	MEANING
0	Successful compilation.
4	Condition code of most severe error was 4.
8	Condition code of most severe error was 8.
12	Condition code of most severe error was 12.
16	Compiler abended due to circumstance detailed in the abend message.

EXAMPLES:

```
$RUN *FORTG
PAR=M
 .
 .      SOURCE MODULES
 .      OPTIONS:SOURCE,MAP,LOAD
 .
 .      OBJECT MODULES
 .
PAR=SML,D
 .
 .      SOURCE MODULES
 .      OPTIONS:SOURCE,MAP,LIST,LOAD,DECK
 .
$ENDFILE
$RUN -LOAD# 5=DATA
```

SOURCE MODULE ERROR/WARNING MESSAGES

The error/warning messages produced by the compiler occur in the source listing immediately following the source statement to which they refer. A maximum of four error messages are printed per line. The following example illustrates the format of these messages as they appear in the source listing.

```
     XX = A+B-C/(X**3-A**-75)
                $          $
   n) y message, n) y message
```

Where: n is an integer noting the positional occurrence of the error on each line.
y is a 1-to-3 digit message number of the form IEYxxxI.
$ is the symbol used by the compiler for flagging the particular error in the statement(this symbol is always noted on the line following the source statement and underneath the error).
message is the actual message printed.

The error and warning messages are distinguished by the resulting completion codes. Serious error messages have a code of eight, while warning messages may produce either a code of four or zero.

IEY001I ILLEGAL TYPE

Explanation: The variable in an assigned go to statement is not an integer variable; or the variable in an assignment statement on the left of the equal sign is of logical type and the expression on the right side does not correspond. CC=8, i.e., The completion code is 8.

IEY002I LABEL

Explanation: The statement in question is unlabelled and follows a transfer of control; the statement therefore cannot be executed. CC=0.

IEY003I NAME LENGTH

Explanation: The name of a variable, common block, name list or subprogram exceeds six characters in length; or two variable names appear in an expression without a separating operation symbol. CC=4.

IEY004I COMMA

Explanation: The comma required in the statement has been omitted. CC=0.

IEY005I ILLEGAL LABEL

Explanation: Illegal usage of a statement; for example, an attempt is made to branch to the label of a format statement. CC=8.

IEY006I DUPLICATE LABEL

Explanation: The label appearing in the label field of a statement has previously been defined for statement. CC=8.

IEY007I ID CONFLICT

Explanation: The name of a variable or subprogram has been used in conflict with the type that was defined for the variable or subprogram in a previous statement. CC=8.

IEY008I ALLOCATION

Explanation: The storage allocation specified by a source module statement cannot be performed because of an inconsistency between the present usage of a variable name and some prior usage of that name. CC=8.

IEY009I ORDER

Explanation: The statements contained in the source module are used in an improper sequence. CC=8.

IEY010I SIZE

Explanation: A number used in the source module does not conform to the legal values for its use. CC=8.

IEY011I UNDIMENSIONED

Explanation: A variable name is used as an array and the variable has not been dimensioned. CC=8.

IEY012I SUBSCRIPT

Explanation: The number of subscripts used in an array reference is either too large or too small for the array.

IEY013I SYNTAX

Explanation: The statement or part of a statement to which this message refers does not conform to the FORTRAN IV syntax. CC=8.

IEY014I CONVERT

Explanation: The mode of the constant used in a DATA or in an Explicit Specification statement is different from the mode of the variable with which it is associated. The constant is then converted in the correct mode. CC=0.

IEY015I NO END CARD

Explanation: The source module does not contain an end statement. CC=0.

IEY016I ILLEGAL STA.

Explanation: The context in which the statement in question has been used is illegal. CC=8.

IEY017I ILLEGAL STA. WRN.

Explanation: A RETURN 1 statement appears in a function subprogram. CC=0.

IEY018I NUMBER ARG

Explanation: The reference to a library subprogram specifies an incorrect number of arguments. CC=4.

IEY019I FUNCTION ENTRIES UNDEFINED

Explanation: If the program being compiled is a function subprogram and there is no scalar with the same name as the function nor is there a definition for each entry, the message appears on SPRINT. A list of the names in error is printed following the message. Although ostensibly a warning diagnostic only, it may lead to unexpected modes being assigned to the undefined entries. If they correspond to initialization entries, then the diagnostic may be ignored. Generally advisable to define all entries completely and properly.

IEY020I COMMON BLOCK NAME ERRORS

Explanation: This message pertains to errors that exist in the definitions of equivalence sets which refer to the common area. The message is produced when there is a contradiction in the allocation specified, a designation to extend the beginning of the common area, or if the assignment of common storage attempts to allocate a variable to a location which does not fall on the appropriate boundary; "name" is the name of the common block in error.

IEY021I UNCLOSED DO LOOPS

Explanation: The message is produced if DO loops are initiated in the source module, but their terminal statements do not exist. A list of the labels which appeared in the DO statements but were not defined follows the printing of the message.

IEY022I UNDEFINED LABELS

Explanation: If any labels are used in the source module but are not defined, this message is produced. A list of the undefined labels appears on the lines following the message. However, if there are no undefined labels, the word NONE appears on the same line as the message.

IEY023I EQUIVALENCE ALLOCATION ERRORS

Explanation: The message is produced when there is a conflict between two equivalence sets, or if there is an incompatible boundary alignment in the equivalence set. The message is followed by a list of the variables which could not be allocated according to the source module specifications.

IEY024I EQUIVALENCE DEFINITION ERRORS

Explanation: This message denotes an error in an equivalence set when an array element is outside the array.

IEY025I DUMMY DIMENSION ERROR

Explanation: If variables specified as dummy array dimensions are not in common and are not global dummy variables, the above error message is produced. A list of the dummy variables which are found in error is printed on the lines following the message.

IEY026I BLOCK DATA PROGRAM ERRORS

Explanation: This message is produced if variables in the source module have been assigned to a program block but have not been defined previously as common. A list of these variables is printed on the lines following the message.

IEY027I CONTINUATION CARDS DELETED

Explanation: More than 19 continuation cards were read for 1 statement. All subsequent lines are skipped until the beginning of the next statement is encountered. The completion code is 0.

IEY032I NULL PROGRAM

Explanation: This message is produced when an end of file mark precedes any true FORTRAN statements in the source module.

IEY033I COMMENTS DELETED

Explanation: More than 30 comment lines were read between the initial lines of 2 consecutive statements. The 31st comment line and all subsequent comment lines are skipped until the beginning of the next statement is encountered. There is no restriction on the number of comment lines preceeding the 1st statement. The completion code is 0.

IEY036I ILLEGAL LABEL WRN

Explanation: The label on this non executable statement has no valid use beyond visual identification and may produce errors in the object module if the same label is the target of a branch type statement. This message is issued, e.g. when an END statement is labelled. The message is a warning only. The completion code is 4.

*FORTH

Description: IBM's FORTRAN H Compiler (at level of OS release 17). The H-level FORTRAN compiler is slower and more expensive to run than the G-level compiler (*FORTG). It does, however, produce object programs that in general take up less storage and run faster. It also produces better error comments, and provides a cross-reference table of symbols used. Note that the major cost of FORTRAN H over G is in loading the compiler. It takes about 11 seconds of CPU time for this. Also the H compiler takes up much more Virtual Memory space: 144 pages as of 2-18-70 . As a result a minimal compilation costs about $1.80 (rates as of 10-17-69), almost all of which is the cost of loading the compiler. It is strongly suggested that if several programs are to be compiled, that they be presented one after the other on SCARDS to the compiler, with an end-of-file only after the last one (this is the so-called "batched" mode). This avoids loading the compiler more than once.

Purpose: To compile FORTRAN IV H source programs.

Logical I/O Units Referenced:

SCARDS - input to the translator. Translator reads FORTRAN programs until an end-of-file is sensed.
SPRINT - printed output from the translator.
SPUNCH - object module produced by translator if DECK option was specified (Default).
0 - object module produced by translator if LOAD option was specified (not default).

Note: if the user does not specify SPUNCH and does specify DECK (default) then the file -LOAD# is created and the object module put into it.

Compiler Options:

The following parameters may be specified (separated by commas) in the PAR= field of the $RUN *FORTH command. Defaults have been chosen to fit the normal usage of FORTRAN H: largish programs where optimization is desired. Description of the parameters follows the list.

PARAMETER_NAME	ABBREVIATION	DEFAULT
BCD EBCDIC	B EB	EBCDIC
DECK NODECK	D NOD	DECK
EDIT NOEDIT	E NOE	NOEDIT
ID NOID	I NOI	NOID
LIST NOLIST	L NOL	NOLIST
LOAD NOLOAD	LO NOLO	NOLOAD
MAP NOMAP	M NOM	MAP
SOURCE NOSOURCE	S NOS	SOURCE
XL NOXL	XL NOXL	NOXL
XREF NOXREF	X NOX	XREF

KEYWORD PARAMETER_NAME	DEFAULT
LINECNT=xx	LINECNT=58
NAME=xxxxxx	NAME=MAIN
OPT={0|1|2}	OPT=2

BCD or EBCDIC

The BCD option indicates that the source module is written in Binary Coded Decimal; EBCDIC indicates Extended Binary Coded Decimal Interchange Code. Intermixing of BCD and EBCDIC in the source module is not allowed.

Notes: 1. If the EBCDIC option is selected, statement numbers passed as arguments must be coded as

&n

However, if the BCD option is selected, statement numbers passed as arguments must be coded as

$n

and the $ must not be used as an alphabetic character in the source module. (The n represents the statement number.)

2. The compiler does not support BCD characters either in literal data or as print control characters. Such characters are treated as EBCDIC. Consequently, a BCD +, for example, used as a carriage control character will not cause printing to continue on the same line. Programs keypunched in BCD, therefore, should be carefully screened if errors relating to literal data and print control characters are to be avoided.

DECK or NODECK

The DECK option specifies that an object module card deck is punched as specified by SPUNCH. NODECK specifies that no object module deck is punched.

EDIT or NOEDIT

The EDIT option specifies that a structured source listing is written to SPRINT. This listing indicates the loop structure and the logical continuity of the source program. If this option is used, OPT=2 must be specified. The NOEDIT option specifies that no structured source listing is written.

ID or NOID

The ID option specifies that the generated code is to contain an identifier for the compiler-assigned internal statement number associated with each function reference or CALL statement. When ID is specified, the internal statement numbers can appear in the diagnostic traceback provided for execution-time errors. The NOID option specifies the omission of the identifiers in the object program.

LINECNT=xx

The LINECNT option specifies the maximum number of lines (xx) per page for a source listing. If LINECNT is not specified, a default of 58 lines per page is provided. (The LINECNT option is effective only at compile time.)

LIST or NOLIST

The LIST option indicates that the object module listing is written to SPRINT. (The statements in the object module listing are in a pseudo assembly language format.) The NOLIST option indicates that no object module listing is written.

LOAD or NOLOAD

The LOAD option indicates that the object module is written on logical I/O unit 0. The NOLOAD option indicates that the object module is not written on logical I/O unit 0.

MAP or NOMAP

The MAP option specifies that a table of names and a table of labels are written to SPRINT. These tables include those names and labels which are generated by the compiler as well as those which appear in the source module. The NOMAP option specifies that no tables are written.

NAME=xxxxxx

The NAME option specifies the name (xxxxxx) assigned to a module (main program only) by the programmer. If NAME is not specified or the main program is not the first module in a compilation, the compiler assumes the name MAIN for the main program. The name of a subprogram is always the name specified in the SUBROUTINE or FUNCTION statement. The name appears in the source listing, map, and object module listing.

OPT={0|1|2}

The OPT=0 option indicates that the compiler uses no optimizing techniques in producing an object module. The OPT=1 option indicates that the compiler treats each source module as a single program loop and optimizes the loop with regard to register allocation and branching. The OPT=2 option indications that the compiler treats each source module as a collection of program loops and optimizes each loop with regard to register allocation, branching, common expression elimination, and replacement of redundant computations.

SOURCE or NOSOURCE

The SOURCE option specifies that the source listing is written to SPRINT. The NOSOURCE option indicates that no source listing is written.

XREF or NOXREF

The XREF option specifies that a cross-reference listing of variables and labels is written to SPRINT. This listing indicates the internal statement number of every statement in which a variable or label is used. The NOXREF option specifies that no cross-reference listing is written.

XL or NOXL

The XL option specifies that the FORTRAN-H extended language features are permitted in the source deck. For details of these features, see the manual, "IBM System/360 Operating System FORTRAN IV (H) Compiler Program Logic Manual", form Y28-6642. The NOXL option specifies that the extended language features are not permitted in the source deck.

Examples:

```
$RUN *FORTH SCARDS=PROGRAM SPUNCH=OBJECT PAR=LIST
$RUN *FORTH
```

EXECUTION TIME CONSIDERATIONS

Input/Output

Logical I/O Units

Although MTS does not provide default assignments for logical I/O units 0 through 9, the FORTRAN IV I/O routines will provide such default assignments during the execution of object modules produced by *FORTG and *FORTH. This defaulting works as follows for object code produced by *FORTG and *FORTH. Any logical I/O unit specified in a FORTRAN IV READ statement will be defaulted to SCARDS unless an explicit assignment is made for the logical I/O unit. Any logical I/O unit specified in a FORTRAN IV WRITE statement will be defaulted to SPRINT unless an explicit assignment is made for the logical I/O unit. These default assignments are made at the time the READ or WRITE is executed, and it is recognized that the pertinent logical I/O unit has no assignment. This default assignment should be used with care, since it is possible through such defaults to have several logical I/O units assigned to the same file or device with intermittent reading and/or writing as a result. One may always use explicit assignments for logical I/O units to prevent any such defaults from occurring.

(Note that due to our terminology we have an illogical situation in the preceding paragraph in which one logical I/O unit is defaulted to another. Actually, it is the FORTRAN IV data set reference number (DSRN) which is defaulted, not the MTS logical I/O unit. That is, when DSRN n is used in a FORTRAN IV READ statement, it is given the same assignment as has been given logical I/O unit n in MTS; if no such assignment has been given for logical I/O unit n in MTS, then DSRN n is assigned to logical I/O unit SCARDS. A similar clarification applies to a DSRN used in a FORTRAN IV WRITE statement. There is, thus, a distinction between the FORTRAN IV DSRNs and the MTS logical I/O units, a distinction which we have ignored by calling each of these items a logical I/O unit. Since in practice this distinction is seldom necessary, we will continue to use the single concept of the logical I/O unit for both of these elements.)

Formatted v.s. Unformatted I/O

Following is a table that displays some comparative times for reading and writing data from/to direct-access files in Fortran, using various formats. The times are in 60ths of a

second and represent the average times to read/write sequentially 50 records, each record corresponding to 25 words of information in processor storage.

Format	Reading	Writing
(25F10.3)	48.6	47.4
(25A4)	12.3	15.3
Unformatted 25 words	7.2	11.6

The table indicates that unformatted I/O is at least 4 times as fast as a typical numeric format, and about 1.5 times as fast as A-format.

Unformatted I/O is best for temporary files containing intermediate results in a processing application.

Formatted I/O has some advantages:

1. File records can be updated externally, i.e. at the MTS command level there are facilities for listing files and modifying them.
2. If the nature of the data allows manipulation via A-format then the time lost in I/O may be offset by the availability of external file modification.

Unformatted files can be handled only by some other program (Fortran or other language), i.e. it does not make sense to "list" such a file (although it may be "copied" into another file). Unformatted data written into a file by a Fortran program can only be read by another Fortran program (or its equivalent); also the corresponding I/O list must be compatible. If a logical record is generated from a list of n words (or memory) during output to a file, then a subsequent input statement attempting to read this record must have a list of $\leq$ n words.

FORTRAN I/O With Subroutines READ and WRITE

READ and WRITE are two assembly language subroutines that are available to the Fortran programmer. Referencing READ or WRITE in a CALL statement in FORTRAN will cause that subroutine to be included when the load module is produced. These subroutines provide the following advantages to the Fortran programmer:

1) Reading and Writing non-blocked, non-formatted records at a much greater rate than that attained using Fortran I/O.
2) Greater versatility in carriage control of tapes at execution time.

The usage of these subroutines is as follows:

CALL{READ / WRITE} (REG,LEN,MOD,LNR,LUNIT,RTN1)

where

REG is an array or variable name indicating the beginning location to or from which data is to be transmitted.

LEN is a half-word integer variable (i.e. INTEGER*2 LEN) indicating the number of bytes to be transmitted. LEN is required by WRITE and is returned by READ.

MOD is a full-word integer which modifies the action of the subroutine when MOD is non-zero and results in a default action when MOD=0. (See Description of Modifiers).

LNR is an (MTS Line Number) *1000 given to or returned from the subroutine depending on whether or not an indexed read or write is indicated by MOD.

LUNIT is the logical unit number corresponding to the file or device on which the operation is to take place.

RTN1 is a Fortran statement label passed to the subroutine. Control is transferred to this statement when READ encounters an end-of-file or when WRITE causes the output device to become full.

Description of Modifiers

The modifiers and their corresponding functions given here are identical to the modifiers which may be attached to the file or device name (e.g. AFILE@CC - see the Files and Devices Manual). The value of MOD will indicate a particular modifier is

1) ON
2) OFF
3) NOT SPECIFIED or BOTH ON and OFF.

If conditions 1 or 2 occur, then the corresponding modifier attached to the file or device name will be overriden. If condition 3 occurs, then either the default modifier is used or, if specified, the corresponding attached modifier to the file or device name is used. The following table indicates the meaning, value and default for MOD.

MODIFIER	VALUE OF MOD		DEFAULT
	OFF	ON	
INDEXED	1	2	OFF
CASECONV	16	32	OFF
CC	64	128	ON PRINTERS/TERMINALS OFF OTHERWISE
PREFIX	256	512	OFF
PEEL	1024	2048	OFF
MCC	4096	8192	OFF
TRIM	16384	32768	ON

To obtain a combination of modifiers, set MOD to the sum of the values of the desired modifiers.

Tape Control Functions

Subroutine WRITE may be used to effect tape carriage control during execution. If the modifier carriage control is ON, then the first three bytes of REG are examined for a control function. A record which is not headed by a valid control function will be written, whereas a valid control function will cause the corresponding control action to be performed and nothing is written. The following table lists the control functions, the corresponding control actions and the meaning and value of a non-zero return code.

CONTROL FUNCTION	ACTION	NON-ZERO RETURN CODE
WTM or WEF	End-of-file written	
FSF	Tape spaced forward past next end-of-file record	4(in end of tape area)
FSR	Tape spaced forward one record	4(tape mark sensed)
BSF	Tape spaced backward past previous end-of-file record	8(load point sensed)
BSR	Tape spaced backward one record	4(tape mark sensed)
REW	Tape rewound to load point	
SRL	Reset maximum record length(length is determined from the 5 characters following 'SRL', left justified with trailing blanks).	4(illegal length)

The indication that an operation by subroutine READ or WRITE was successful, or that special conditions were encountered during an operation, is communicated to the monitoring system (MTS) by a value generated by the subroutine known as the RETURN CODE. If the return code is greater than 8 an error message is printed and execution is terminated.

RETURN CODE	MEANING
0	normal return
4	END-OF-FILE (subroutine READ); END-OF-TAPE (subroutine WRITE);
8	tape loadpoint sensed on backspace command
12	user attempted to write more than 5 records in end-of-tape area.
16	permanent read/write error (tape positioned past bad record on read).
20	attempt to write on a file protected tape.
24	equipment malfunction

Restrictions

1. LEN must be a halfword integer variable.
2. If the output device is a file, an attempt to write more than 255 characters (i.e. LEN > 255) will result in the first LEN (modulo 256) characters being transmitted. If the output device is a tape and LEN > S, where S is the buffer size, the first S characters are written and the remaining characters are lost.
3. Carriage control operations may be carried out with subroutine READ in place of subroutine WRITE, thereby affording protection of the tape during control operations.
   Control operations may be performed on file protected tapes.
4. Card images of $END and $ENDFILE will be recognized from logical units assigned to *SOURCE* or *MSOURCE* exclusively.
5. For users who know what a File or Device Usage Block (FDUB) pointer is; LUNIT may be the location of a FDUB pointer.

Timing

Operations with READ and WRITE are approximately 2.8 times as fast as the corresponding Fortran operations.

Examples

1. Subroutine READ

To obtain one input record from a specified logical unit the user must provide values for MOD, LUNIT, RTN1. The value given to MOD may indicate an indexed read in which case the user must also give a value for LNR.

```
      DIMENSION A(64)
      INTEGER*2   LEN
        .
        .
        .
      MOD=0
      LUNIT=4
      CALL READ (A,LEN,MOD,LNR,LUNIT,&100)
        .
        .
        .
100   PRINT 1,LUNIT
1     FORMAT('END OF FILE ENCOUNTERED ON UNIT',I4)
        .
        .
        .
```

In the above example, a record would be read from the file corresponding to logical unit 4 and transmitted to A. The number of characters transmitted would be placed in LEN and the record index of the line read would be returned in LNR.

Note that the array A is large enough to contain the maximum size record that can be present in a file (i.e. 255 characters). Similar precautions with the size of A would have to be taken if the input device is not a file.

2. Subroutine WRITE

To write one output record on a specified logical unit, the user must provide values for LEN, MOD, LUNIT and RTN1. The user must also give a value for LNR if an indexed modifier is in effect (see Description of Modifiers).

```
      DIMENSION A(8)
      INTEGER*2 LEN
        .
        .
        .
      LEN=32
      CALL WRITE (A,LEN,2,4100,6,&200)
        .
        .
        .
200   PRINT 1
1     FORMAT ('OUTPUT DEVICE IS FULL')
        .
        .
        .
```

In the above example, the contents of the array A would be written in line number 4.1 (record index 4100) of the file corresponding to logical unit 6.

Note that MOD=2 produces an indexed operation, so it was necessary to supply a record index.

3. Tape Control

Space tape forward past next end-of-file record on logical unit 4.

```
      INTEGER*2 LEN
        .
        .
        .
      LEN=3
      CALL WRITE('FSF',LEN,128,LNR,4,&300)
        .
        .
        .
300   PRINT
1     FORMAT('TAPE SPACED FORWARD INTO END-OF-TAPE AREA')
        .
        .
        .
```

Note that MOD = 128 turns CARRIAGECONTROL (CC) on.

Subroutine Libraries

*LIBRARY

Contents: FORTRAN IV Library Subroutines and MTS Application Subroutines.

Usage: *LIBRARY is automatically searched when the $RUN command is issued and any FORTRAN IV Library Subroutine or MTS Application Subroutines referenced are included with the fortran object module when the load module is produced.

Description: For a description of the FORTRAN IV Library Subroutines, consult the IBM System/360 Operating System: FORTRAN IV Library Subprograms, Form C28-6596. The MTS Application Subroutines are described in the MTS LIBRARIES MANUAL. In addition to *LIBRARY, there are a number of System Subroutines that are availabe to the FORTRAN programmer. These subroutines are also described in the MTS LIBRARIES MANUAL.

*SSPLIB

Contents: *SSPLIB contains exactly one line, namely

```
$CONTINUE WITH *SSPMATHLIB+*SSPSTATLIB
```

It invokes the two sections of the Scientific Subroutine Package, the mathematical and statistical routines, repectively.

Usage: The user should concatenate *SSPLIB to the file containing his object modules.

Example:

```
$RUN -OBJ+*SSPLIB 5=*SOURCE*
```

Description: The subroutines contained in the SSP library are fully documented in the IBM publication System/360 Scientific Subroutine Package, (360A-CM-03) Version III, Programmer's Manual (H20-0205). See also descriptions of *SSPMATHLIB, *SSPSTATLIB, *SSPSOURCE, and *LIBRARY in this Manual.

Some of the functions performed by SSP modules are duplicated by subroutines available in *LIBRARY. In general the *LIBRARY versions are both faster and smaller and are therefore highly recommended for use in preference to the corresponding SSP routines. See *LIBRARY for description of these subroutines.

*SSPSTATLIB

Contents:	*SSPSTATLIB contains the statistical routines of the System/360 Scientific Subroutine Package, SSP.
Usage:	The user should concatenate *SSPSTATLIB to the file containing his object modules.
Example:	$RUN -OBJ+*SSPSTATLIB 5=*SOURCE*
Description:	See the Description of *SSPLIB in this section.

*SSPMATHLIB

Contents:	*SSPMATHLIB contains the mathematical routines of the System/360 Scientific Subroutine Package, SSP.
Usage:	The user should concatenate *SSPMATHLIB to the file containing his object modules.
Example:	$RUN -OBJ+*SSPMATHLIB 5=*SOURCE*
Description:	See the Description of *SSPLIB in this section.

Execution Time Errors and Diagnostics

Subroutines GETIHC and PUTIHC

When one of the IHC-type error conditions occur, its number is put in a variable named in the last GETIHC call; then, if its number is found in a vector named in the call, control is sent (via implicit RETURNS if necessary) to a statement named in the call; otherwise the standard error action is taken. PUTIHC restores standard error processing.

The variable and vector elements are integer of length 4. The first element in the vector is the number of error codes that follow (order is not important among the error codes), unless this count exceeds 39, when all IHC errors are returned, and the rest of the vector is ignored. The vector may be modified after the call, the modified list being inspected after the ensuing IHC error. A call on these routines takes about 10 microseconds.

Examples:

```
     i) CALL PUTIHC
    ii) CALL GETIHC (IERR,LIST,&100)
   iii) If LIST in ii) were specified by
          INTEGER LIST(3)/2,251,215/
```

control would be returned for error codes 251 and 215 i.e. for negative SQRT arguments, and illegal characters in numeric fields.

*IHC

Contents: Error messages for the error numbers associated with the error messages produced by the Fortran library subroutines.

Usage: Line xxx of *IHC contains the description of error mumber IHCxxx. Hence given "IHC232I", then doing

```
$COPY  *IHC(232,232)
```

will bring forth the explanation for that error message on his terminal.

Execution Time Error Messages

IHC	CAUSE
211	IBCOM-UNDEFINED FORMAT SPECIFICATION
212	IBCOM-FORMAT EXCEEDS RECORD LENGTH
213	IBCOM-I/O LIST EXCEEDS RECORD LENGTH
214	DEFINE FILE STMT NOT SUPPORTED IN MTS
215	FCVTH-INVALID CHARACTER IN NUMERIC FIELD
225	FCVTH-INVALID HEX CHARACTERS
221	NAMELIST-VARIABLE NAME TOO LONG
222	NAMELIST-VARIABLE NAME NOT IN DICTIONARY
223	NAMELIST-VARIABLE NAME HAS NO DELIMITER
224	NAMELIST-SUBSCRIPT ON SCALAR VARIABLE
241	INTEGER 0 ** NON-POSITIVE INTEGER
242	REAL*4 0 ** NON-POSITIVE INTEGER
243	REAL*8 0 ** NON-POSITIVE INTEGER
244	REAL*4 0 ** NON-POSITIVE REAL*4
245	REAL*8 0 ** NON-POSITIVE REAL*8
246	COMPLEX*8 0 ** NON-POSITIVE INTEGER
247	COMPLEX*16 0 ** NON-POSITIVE INTEGER
251	SORT AGRUMENT NEGATIVE
252	EXP ARGUMENT > 174.673
253	LOG ARGUMENT NON-POSITIVE
254	COS/SIN ABS(ARGUMENT) > 2**18*PI (8E+5)
259	TAN/COTAN ARGUMENT CLOSE TO SINGULARITY
261	DSQRT ARGUMENT NEGATIVE
262	DEXP ARGUMENT > 174.673
263	DLOG ARGUMENT NON-POSITIVE
264	DCOS/DSIN ABS(ARGUMENT) > 2**50*PI
265	BOTH ARGUMENTS TO DATN2 ZERO
266	DSINH/DCOSH ABS(ARGUMENT) > 174.673
267	DARSIN/DARCOS ABS(ARGUMENT) > 1
268	DTAN/DCOTAN ABS(ARGUMENT) > 2**50**PI
269	DTAN/DCOTAN ARG CLOSE TO SINGULARITY
271	REAL PART OF CEXP ARGUMENT > 174.673
272	CEXP ABS(IMAG(ARG)) >=2**18*PI
273	CLOG ARGUMENT ZERO
274	CSIN/CCOS ABS(REAL(ARG)) >= 2**18*PI
275	CSIN/CCOS ABS(IMAG(ARG)) > 174.673
281	CDEXP REAL(ARGUMENT) > 174.673
282	CDEXP ABS(IMAG(ARG)) >= 2**50*PI
283	CDLOG ARGUMENT ZERO
284	CDSIN/CDCOS ABS(REAL(ARG)) >= 2**50PI
285	CDSIN/CDCOS ABS(IMAG(ARG)) > 174.673
290	GAMMA RANGE IS 2**-252 < ARG < 57.5744
291	ALGAMA RANGE IS 0 < ARG < 4.2937E+73
300	DGAMMA RANGE IS 2**-252 < ARG < 57.5744
301	DLGAME RANGE IS 0 < ARG < 4.2937E+73

MISCELLANEOUS

*SSPSOURCE

Contents: The object module of a program which allows the user to obtain copies of FORTRAN IV source decks of SSP subroutines and/or example programs.

Usage: The program is invoked by a $RUN command specifying *SSPSOURCE as the object file with an optional parameter specifying whether the user desires prompting messages or not.

Examples:
```
$RUN  *SSPSOURCE  SCARDS=ZZ  PAR=QUICK
      (no prompting)

$RUN  *SSPSOURCE
      (with prompting)
```

Logical I/O Units referenced:

SCARDS - For the FDname of where the source modules should be written, followed by the deck IDs of the desired source modules and the page numbers of the desired example programs.

SPRINT - Prompting messages and verification of the input for conversational users unless PAR=QUICK is specified.

(The program also mounts the SSP distribution tape with pseudodevice name *SRC.SSP* to obtain the modules. Comments are printed on SPRINT to inform the user of the delay necessary for the tape mounting.)

Description: The specific prompting comments given conversational users are self-explanatory and will not be given here.

The first input line should be the file or device name of the place to put the selected source modules. The name is terminated by the first blank and accordingly should be left-justified. A zero length line or an end-of-file will cause program termination. If connection of the FDname is not possible, a batch job will be terminated while a conversational job will read SCARDS for a new FDname. This situation is flagged with the comment

```
FILE/DEVICE BUSY OR NOT OPERATIONAL.
```

When the output FDname is successfully connected, it is rewound if it corresponds to a tape or file. Subsequently, when all source modules have been obtained, an end-of-file and rewind are issued if the FDname specified a tape. Subsequent input lines, up to the first zero length line or end-of-file, specify the deck ID for the SSP subroutines desired and/or the page number for those example programs

desired. This information is available from the SSP Programmer's Manual (H20-0205). The four character deck identifications may be obtained either by looking at the source listing given in this Manual or from The Alphabetic Guide to Subroutines, pp. 19-26. The page numbers for the example programs are those given in the Categorical Guide to Subroutines, pp. 17-19. Note that the IDs "LOC " and " LOC" are not the same; the former corresponds to the SSP subroutine LOC, while the latter would occasion the comment

DECK LOC NON-EXISTENT.

The SCARDS input lines are read according to a multiple A4 format with the first completely blank four character field terminating the input for that line. Within each four character field, all input should be left-justified with a trailing blank if necessary. Two special ID's, FIL1 and FIL2, are available to obtain routines TALLY through HSBG (15380 cards) and PADD through DJEL (25113 cards), respectively. Deblocked, this combination exceeds a full tape reel.

When all the input has been read, it will be summarized on SPRINT in the form

C DECK ----

or

C EXAMPLE PROGRAM FROM P--- .

whichever is appropriate. Identical lines will also be inserted in the output file to separate the various source modules requested, hence the C in column 1. Note that the example programs will not compile directly since they contain a /DATA card and the numeric test data. Aside from the separator cards, the output consists of 80-character card images.

If, after this entire input procedure, all copies of the SSP distribution tape are in use or all 9-track tape drives are in use, the program will terminate with the comment

SSP TAPE OR TAPE UNIT UNAVAILABLE.

Examples: In the conversational examples which follow the lines typed in upper case are program or system generated while those in lower case represent terminal input.

```
#$run  *sspsource
#EXECUTION BEGINS
SSP SOURCE SNATCH PROGRAM.
ENTER OUTPUT FDNAME.
list
ENTER DECK ID OF DESIRED ROUTINE
OR PAGE NUMBER OF EXAMPLE PROGRAM.
smo gels
sel3439 del3
```

```
(carriage return)
C DECK  SMO
C DECK  GELS
C DECK  SE13
C DECK  DE13
C EXAMPLE PROGRAM FROM P439 .
MOUNTING *SRC.SSP*
DONE
DONE
DISMOUNTING *SRC.SSP*
#EXECUTION TERMINATED
```

To illustrate the effect of the optional parameter,

```
#$run *sspsource par=quick
#EXECUTION BEGINS
list
aratfratcnpsapllapfs      ,
(carriage return)
MOUNTING *SRC.SSP*
etc.
```

Finally, to obtain the subroutine CANOR and all subroutines that it calls upon one might do the following with a card estimate of at least 718.

```
$RUN *SSPSOURCE
*PUNCH*
CANOMINVNROOEIGE
$ENDFILE
```

It should be noted that among the comment cards at the beginning of each source module is a section entitled:
SUBROUTINES AND FUNCTION SUBPROGRAMS REQUIRED.

*FORTEDIT

Contents: The object module of a program to convert FORTRAN statements in free-format form to fixed-format form.

Purpose: To allow the user to write free-format FORTRAN statements. The output consists of "fixed-format cards" which can be processed by the current FORTRAN compilers. If input is being entered from a terminal, a free-form to fixed-form conversion can be done as the lines are typed in by setting logical tabs and using them. See the appropriate Terminal Usage Guides in Volume 1 of the MTS Manual.

Usage: The program is invoked by the appropriate $RUN command specifying *FORTEDIT as the file where the object cards are found.

Logical I/O Units Referenced:

SCARDS - free-format input lines to be converted.
SPUNCH - fixed-format output lines for processing by the FORTRAN IV compilers.

Example:

```
$RUN *FORTEDIT SCARDS=FTNPRGM SPUNCH=-PROG
```

Description:

Prefixing

When SCARDS does not refer to a line file, the program attempts to prefix each input line with the current line number associated with SPUNCH. This helps the user to see where the output line will go into SPUNCH, particularly when the line number appears on the terminal. The user should bear in mind that due to the possibility of continuation lines, the line number is not always constantly incremented but is nevertheless accurately updated for the next line.

Initial lines

These lines may start with a statement number which can be preceded by any number of blanks or tabs. The statement number must be no longer than five digits and must be followed by a blank or a tab. If a line is to be continued, the user should type a hyphen and then terminate the line. The hyphen is, of course, not considered part of the text.

Continuation lines

These are the lines which follow the continued lines. The text is continued from position 1 of the input line.

Comment lines

These lines can be identified by the letter "C" in the first typeable position. Of course, the line C=3.14 beginning in the first position will be treated as a comment line. If the line is intended to be a statement, the user should tab or space before typing a "C". As a safeguard, the user may set tab stops and use tabs since the program will treat tabs as blanks. This has the advantage that input lines will be easily read.

Inserting a line

To insert a line in an output line file, the user should first type the desired line number followed by a comma. The text may then be typed. If any continuation lines are required, the line number will be automatically incremented by the current increment.

Command lines

Any non-continuation line with an "*" in the first position is a command line. Command lines perform functions such as deleting a line or printing the output file. The command lines are applicable only when SPUNCH refers to an MTS line file. The user need only type the first character of a command name preceded by the control character "*", i.e., *L for *LIST .

Emptying the file

The command line *EMPTY prompts the user for a confirmation. If confirmation is given, the line file is emptied.

Numbering the file

The general format for numbering is:

```
*NUMBER b,e,i
```

where b is the beginning line number, e is the ending line number, and i is the increment. The default for the *NUMBER command is 1,99999.999,1. In addition, if the user previously typed

```
$RUN *FORTEDIT SPUNCH=-X(12,45,1)
```

the program issues an automatic command: *NUMBER 12,45,1. In particular when the output line number is greater than e, the user will get a comment and the program is automatically terminated.

Listing the file

The command *LIST is used to print the lines of the output file. The format for *LIST is the following:

```
*LIST a,b
```

where a and b are line numbers. The defaults for a and b are the beginning line number and the end line number. If the user wants to list a file, he may type: *LIST 10.00,+20.

Copying the file

The command *COPY is identical to *LIST except line numbers are not printed.

Deleting a line

If one line is to be deleted, the user should type

```
*DELETE I
```

where I is the line number of that line. The command

```
*DELETE -10,20.5
```

can be typed to delete lines from -10 to 20.5.

Setting attention

The user may issue an attention interrupt (during the listing of the output file, for instance). *FORTEDIT will ask whether the user desires the program to be terminated. If the user types NO, then the program resets the output line pointer and is ready to accept the next input line.

*TIDY

Contents: The object module of TIDY, a program which renumbers and edits FORTRAN source programs.

Usage: *TIDY is invoked by a $RUN command specifying it as the file where the object module is located.

Logical I/O Units Referenced:

```
SCARDS - Fortran source input
SPUNCH - altered Fortran source input
1      - scratch file
2      - scratch file
6      - listing of old and new Fortran source
```

Example:

```
$RUN *TIDY SCARDS=FORTPROG
       SPRINT=LISTING SPUNCH=PUNCH 1=-T1 2=-T2

$RUN *TIDY 1=-T1 2=-T2
     ┌          ┐
     | FORTRAN  |
     | program  |
     | with TIDY|
     | control  |
     | cards    |
     └          ┘
*LAST
*STOP
$ENDFILE
```

Description: TIDY is a FORTRAN program that renumbers and edits other FORTRAN source programs whose statement numbering has become unwieldly and whose readability has deteriorated as a result of the many revisions, patches, and corrections that are typical of reworked programs. TIDY processes programs routine-by-routine and punches new versions of the programs with the following characteristics: (1) all statement numbers increase in consecutive order; (2) only statements referred to by other statements retain statement numbers; (3) all statement number references are updated to conform to the numbering scheme; (4) all FORMAT statements are collected and appear at the end of each routine; (5) all FORMAT and CONTINUE statements that are not referenced are deleted; (6) blanks are interspersed in the FORTRAN statements to improve the readability of the statements, while excessive blanks in the statements are deleted; (7) comments are processed to delete excessive blank comments and to eliminate comments from the FORTRAN statement number and continuation fields;

and (8) each card is labeled with a unique letter-number combination. TIDY is entirely written in ASA FORTRAN and accepts and processes all ASA FORTRAN statements as well as some IBM and CDC dialect statements.

Since TIDY will convert FORTRAN-II I/O statements into their FORTRAN-IV equivalent, this program can be used as an aid in the conversion of FORTRAN-II programs to FORTRAN-IV.

Warnings: As it currently exists, *TIDY does the following unfortunate things:

(1) *TIDY deletes literal blanks in format statements.

(2) *TIDY deletes "IMPLICIT" statements

i.e., IMPLICIT REAL*8 (A-Z)

or

IMPLICIT INTEGER (A-K,Q-Z)

would be deleted from the deck.

(3) *TIDY deletes I/O statements which contain an end of file exit

i.e., READ(5,100,END=999) A,B,C

would be deleted from the deck.

Table I

TIDY CONTROL CODES

Code Word	Example of Word Use
*BASE	*BASE = 100.
*NOBASE	*NO BASE
*CARD	*CARDS
*NOCARD	*NO CARDS
*COLL	*COLLECT FORMATS
*NOCOLL	*NO COLLECTION
*COMM	*COMMENTS
*NOCOMM	*NO COMMENTS
*EXEM	*EXEMPT
*NOEXEM	*NO EXEMPTIONS
*IDIN	*ID INCREMENT = 2.
*IDST	*ID STEP = 2.
*LABE	*LABEL
*NOLABE	*NO LABEL
*LAST	*LAST
*LIST	*LIST
*NOLIST	*NOLIST
*NEWR	*NEW ROUTINE
*REFE	*REFERENCES
*NOREFE	*NO REFERENCES
*ROUT	*ROUTINE = 5.
*SKIP	*SKIP
*STAT	*STATEMENT INCREMENT = 10.
*STOP	*STOP

The individual TIDY control cards are discussed below. Preceding each discussion, is an example of the control statement, with the required portion of the control word underlined. In addition, those codes that are immediate in action are labeled "(IMMEDIATE)," while those deferred until after the first pass are labeled "(DEFERRED)."

*BASE = 100. (DEFERRED)

Although TIDY normally starts the statement numbering with "1" and the increments the statement numbers uniformly (e.g., 1, 2, 3, ... etc., or 5, 10, 15, 20, ... etc.), a base number may be added to all statement numbers by the use of the *BASE control statement. In this case, the base is the number given between the "=" sign and the period. For the example given above, TIDY might assign statement numbers in the following order: "101, 102, 103, ... etc."

*CARDS (DEFERRED)
*NOCARDS

Normally, TIDY punches the TIDY-processed version of the FORTRAN program. The *NOCARDS control code inhibits such punching. Similarly, *CARDS restores normal punching. If TIDY discovers a serious blunder in a FORTRAN source deck, card-punching will be automatically suppressed, although TIDY will go on to print a listing of the TIDY-processed faulty deck. The *CARDS control code will override the punching suppression if it appears in the source deck after a card containing such a blunder.

*COLLECT (IMMEDIATE)
*NOCOLLECT (IMMEDIATE

Normally, TIDY collects all FORMAT statements encountered within each routine and places them at the end of the routine. This collection may be suspendad by the use of the *NOCOLLECT control word. Collection may be restored by the use of the *COLLECT control code. FORMAT statements bracketed by *NOCOLLECT and *COLLECT control cards will be left in their position in the FORTRAN routine, while others will be collected normally and will appear at the end of the routine.

*COMMENTS (DEFERRED)
*NOCOMMENTS (DEFERRED)

Although TIDY normally processes FORTRAN comments and puts them in the updated version, the *NOCOMMENTS control option may be used to inhibit the output of all comments. The option is cancelled by the *COMMENTS control code. Because these control options are deferred, bracketing comments to delete them cannot be done. The last *COMMENTS or *NOCOMMENTS control statement will apply.

*EXEMPT (DEFERRED)
*NOEXEMPT (DEFERRED)

Certain programmers like to punch FORTRAN statements, such as COMMON or DIMENSION statements, in special ways so that names line up in columns, and so forth. Normally, TIDY, in its attempt to eliminate unessential blanks in FORTRAN statements, will destroy such column arrangements. The *EXEMPT control code causes TIDY to exempt all nonexecutable statements from internal processing and, hence,from elimination of blanks. The *NOEXEMPT control code restores normal processing. If only certain statements in a given routine need protection, they may be bracketed by a pair of *EXEMPT and *NOEXEMPT control words. Note that statements protected from internal processing by TIDY are also protected from diagnostic checking by TIDY and may contain errors in the output that normally would have been detected by TIDY.

* ID STEP = 2. (DEFERRED)
* ID INCREMENT = 2. (DEFERRED)

These control statements set the card sequence increment so that the identification columns (columns 76 through 79) of the output card will be labelled in multiples of the increment given between the "=" sign and the period. In the above examples, TIDY might label the output cards: "2, 4, 6, 8, ... etc." A negative or zero value given in these control statements is interpreted by TIDY as a normal increment of 1. Care must be taken not to assign too large an increment value since the maximum label number is "9999."

*LABEL (DEFERRED)
*NOLABEL (DEFERRED)

The first card of each routine has card columns 73, 74 and 75 inspected by TIDY for possible use in labeling the revised deck. However, TIDY normally does not use these columns. Instead, TIDY uses a unique letter-number combination to identify cards in a routine.

For example, all cards belonging to the first routine of a program will normally be labelled with the letter "A" in column 75. Within this routine the cards will be sequentially numbered in columns 76, 77, 78, and 79. The next routine will normally be labelled with the letter "B" in column 75 and will be internally numbered. The following routine will be routine "C", and so forth.

The *LABEL control card causes TIDY to suppress the normal alphabetical labelling, and to label each routine with the contents of columns 73, 74, and 75 of the first card of that routine. Note that the label does not come from the *LABEL card itself unless the *LABEL card is the first card of the routine on which it appears. Ordinarily, the programmer will put the label that he wishes on the first card of his routine (e.g., the SUBROUTINE, FUNCTION, or PROGRAM statement), and then follow that card with the *LABEL control card. Of course, if columns 73, 74, and 75 of the first card are blank, then the output cards will be blank in these columns and only sequentially numbered in columns 76 through 79.

The *NOLABEL control statement cancels the labelling option and restores normal alphabetic labelling. TIDY keeps count of routines being processed so that, if labelling is restored, the letter of the alphabet corresponding to that routine number is used. For example, if normal labelling is restored in the fifth routine, that routine will be labelled with "E" in column 75 . Concern need not be wasted on programs containing more than 26 routines since TIDY has provisions for generating 676 unique one- and two-letter alphabetic labels.

*LAST (IMMEDIATE)

This control, which is identical in action to *STOP, calls for an immediate termination of TIDY, and is used to indicate that the last routine has been processed. Because TIDY uses an input buffering routine that reads one card ahead of the statement being processed, both a *LAST and a *STOP card should follow the last END statement of the input FORTRAN deck. If both cards are not used, the computer system monitor may abort TIDY abnormally because of an input "END-OF-FILE."

Note that *LAST and *STOP should not appear in a routine to be processed since they cause an immediate stop, and therefore will not permit a partially processed routine to finish processing.

*LIST (IMMEDIATE)
*NOLIST (IMMEDIATE)

TIDY normally prints two lists of each routine; one showing the original version of the routine and one showing the TIDY-processed version of the same routine. The listing of the original routine may be suppressed by the use of the *NOLIST control statement. Only the final version of the TIDY-processed routines will then be listed. Since listing suppression may be desirable for reasons of economy in the case of well-documented programs or for routines for which no TIDY diagnostic comments are anticipated.

The first pass listing is restored by the *LIST control statement, by the generation of a TIDY diagnostic, or by a request for a reference directory (*REFERENCES).

The use of the *NOLIST option will, in general, result in a substantial saving in run time, since much less printing is done under this option.

*NEWROUTINE (DEFERRED)

This control card resets the routine counter so that--if normal alphabetic labelling is done--the current routine being proceseed will be labeled "A."

The *NEWROUTINE also restores all TIDY flags to their normal mode, as if TIDY were starting anew. The *NEWROUTINE statement implies the following

option setup: *BASE = 0., *CARDS, *COLLECT, *COMMENTS, *NOEXEMPTIONS, *ID INCREMENT = 1., *NOLABELS, *LIST, *NOREFERENCES, and *STATEMENT INCREMENT = 1. If any of these options are not as desired following the execution of *NEWROUTINE, they may be reset by the use of the appropriate control cards.

*REFERENCEES (DEFERRED)
*NOREFERENCES (DEFERRED)

The programmer may call for an optional statement number reference directory through the use of the *REFERENCES control statement. The reference directory lists all new statements numbers, the corresponding old statement numbers, and the location of the statements in the old routine. Since the reference directory would be of little use without a listing of the original routines, the use of *REFERENCES implies the execution of a *LIST command.

The reference directory is useful for debugging or tracing the logical flow of very long and complicated routines. A reference directory is not prepared for routines having fewer than five numbered statements in the TIDY-processed version.

The *NOREFERENCES statement inhibits the preparation of the directory.

*ROUTINE = 26. (DEFERRED)

This control statement sets the routine counter to the number between the "=" sign and the period. If the normal alphabetic labelling option is on, the current routine will be labelled with the letter (or letters) corresponding to the number given (i.e., "Z" for the above example). The use of *ROUTINE does not change any other control card options.

*SKIP (IMMEDIATE)

This control card causes TIDY to skip processing of the routine in which it appears. Instead, TIDY will execute a simple list of the routine until the END card is found. No

internal processing other than the test for the END statement is done. The routine being skipped is not punched. The routine counter, however is incremented.

Upon encountering the END statement, the *SKIP option is cancelled and TIDY resumes normal processing starting with the following card.

*STATEMENT NUMBER INCREMENT = 2. (DEFERRED)

This control statement causes TIDY to increment statement numbers by the value given between the "=" sign and the period. For the above example, TIDY would number statements in the following sequence: "2, 4, 6, ..., etc." A negative or zero number in the *STAT control statement results in the normal increment value of 1 being assigned.

*STOP (IMMEDIATE)

This control statement, like *LAST, causes immediate termination of the execution of TIDY. For further comments, see the description of the *LAST control statement.