SPECTRA 70

RADIO CORPORATION OF AMERICA • ELECTRONIC DATA PROCESSING

SYSTEM 70/45

BASIC TIME SHARING SYSTEM REFERENCE MANUAL

RADIO CORPORATION OF AMERICA

70-45-601

July 1967

The information contained herein is subject to change without notice. Revisions may be issued to advise of such changes and/or additions.

First Printing: July, 1967

CONTENTS

CONTENTS (Cont'd)

1. INTRODUCTION

◆ The RCA 70/45 Basic Time Sharing System (BTSS) is a programming system that provides concurrent computer access to a maximum of 16 remotely located users. The programming language employed by the user is interactive FORTRAN and is supplemented by a set of operating statements. The user communicates with the system either in a conversational manner in which his input is processed one statement at a time, or in a batch manner in which a complete program is executed at a later time. The typical equipment configuration is as follows:

Typical Configuration

1	70/45F	Processor (131K) with Multiplexor Channel Memory Protect Feature and Elapsed Time Clock Feature
1	5016	Selector Channels (3 channels)
1	70/97	Console and Typewriter
1	70/234	Card Punch
1	70/237-10	Card Reader
1	70/242-10	Printer
3	70/432	Magnetic Tape Units
1	70/472-208	Magnetic Tape Controller
2	70/551	Random Access Controller
3	70/564	Disc Storage Units
6	70/563	Disc Packs
1	70/668-11	Communications Controller Multichannel
1-16	70/720	Communication Buffers
1-16	M33/M35	Teletype Terminal(s)
1-4	6050-12	Video Terminal(s)
1-16	103-A-2	Data Sets

If a remote user requires a batch type process, the information control parameters and data for this process are stored temporarily on a disc. This information is transcribed to a job stream tape for subsequent processing when the computer is operating in a batch mode under control of the Tape Operating System.

The Basic Time Sharing System fulfills the following user requirements:

1. The remote user is provided with a problem-oriented language for control of the system. All displays of control information and debugging data are consistent with the systems' programming language; that is, the user has compatible languages for programming and communicating with the system.

1. INTRODUCTION (Cont'd)

2. The user has the ability to stop his remote terminal at any time without loss of data. Such simple tasks as paper and ribbon changing or discontinuing a job dictate this requirement.

3. The remote user can modify data without complete retransmission, and has the option to list and inspect data selectively, rather than transmitting entire output files. The user has the ability to keep his files in random storage from which they may be retrieved when the need arises.

In all instances the user has the impression that he is the sole user of the system and that all facilities are under his control. Specifically, the user is secure from unwarranted action by other users. In conjunction with the above, the user is able to start his jobs without having to wait and to continue using the system as long as the system is in a Time Sharing mode.

These user requirements have been solved in the following ways: The source language, command language, and output data are designed in a problem-oriented fashion for ease of learning and use. Output diagnostic messages are explicit and allow debugging on the same level as program construction. A facility to execute programs, modify or correct programs and data, and to request data selectively is included within the system complex.

2. GETTING ON THE SYSTEM

GETTING ON-LINE WITH BTSS

◆ The Model 35 Teletype Unit and Model 70/6050 Video Data Terminal are the remote devices supported by BTSS. The following procedure allows the user of a Model 35 Teletype to get on-line:

1. Press ORIG.
2. Listen for dial-tone (turn up speaker volume control).
3. Dial number.
4. When phone is answered by the system a high-pitched tone occurs followed by the typeout BEGIN (you have only 50 seconds to log on from this point).
5. Press keyboard (K) button in lower left of console.
6. Log on by typing:

   /ONΔ(eight character user code assignment)
7. Press RETURN button.
8. The system responds with READY.

NOTE

If either log on takes more than 50 seconds or a wrong user code is entered the terminal is disconnected and all of the previous procedures must be repeated with the addition of pressing the CLR button before ORIG.

The following procedure allows the user of a 70/6050 Video Data Terminal to get on-line:

1. Log on by typing /ON usercode.
2. When the complete message is typed, press the SHIFT key and the M key. This displays a closed bracket symbol that indicates the end of message.
3. Press TALK button on the DATA-PHONE set.
4. Pick up receiver; when dial tone occurs, dial number.
5. Listen for high pitch tone.
6. When tone is heard, press DATA button and hang up.
7. Press the transmit key.
8. The system responds with BEGIN.

Notes

◆ 1. For a detailed description of the Model 35 Teletype unit refer to Teletype Operating Manual.

2. For a detailed description of the Model 6050 Video Data Terminal refer to RCA Video Data Terminal Operating Manual.

INPUT REQUIREMENTS

All Input

◆ 1. Must be less than 256 characters long.

2. Must not begin before the system has responded to the previous input. The minimum response is a line feed.

3. Must be followed by a carriage return (or] in the case of the 6050).

Paper Tape Input

◆ 1. Must be followed by the three characters:

X-OFF

Line Feed (Or any character except RUB-OUT)

Carriage Return

2. When one tape input has been read and processed, the next one will automatically be read unless an error or stop condition occurred during processing.

DOCUMENTATION CONVENTIONS

◆ The illustrations in this manual of command and statement formats make use of braces or brackets with information enclosed between them. Braces { } denote that a choice must be made from the two or more entries enclosed. Brackets [ ] signify optional information that may be employed, or omitted, as the user requires.

THE SYSTEM RESPONSES ILLUSTRATED IN THE EXAMPLES THROUGHOUT THE MANUAL ARE DISPLAYED IN AN ITALIC TYPEFACE.

3. COMMAND LANGUAGE

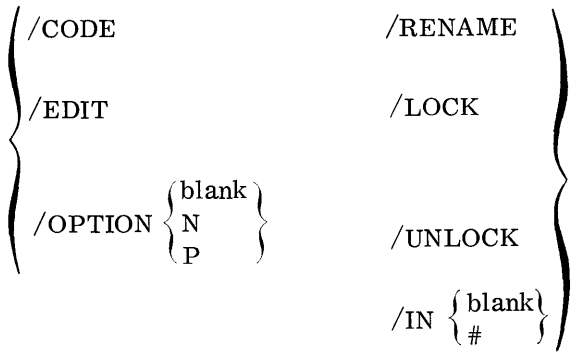
COMMAND LANGUAGE DESCRIPTION

◆ The interactive user communicates with the Basic Time Sharing System by entering a string of characters terminated by an end-of-message character. The system always acknowledges receipt of these character strings, called messages, by a visible response at the console. Most responses are in the form of a typeout, however, some responses are a line feed only. A message may be data or a command to the system, and may contain control characters, if desired. A system command is distinguished by its format; that is, the message begins with one virgule symbol (/), followed by the command name. Messages with any other format are considered to be data. When the first two characters in any message are virgule symbols (//), the first symbol is discarded and the remaining characters are treated as data. This convention allows the virgule symbol to be handled as a legitimate character in any character set.

Also, because most system commands have associated with them an optional sequence field or a filename the following definitions apply throughout the system:

1. A sequence number (seq-#) must be between one and six characters long and must be a number between 0 and 999999.

2. When a user creates a file he assigns it a name which is used thereafter to retrieve the file. The filename is limited in length to eight characters. At least one of the filename characters must be nonnumeric and none may be blank. The following commands may be employed by the user to create files:



The remote user must also be aware of the method employed to allocate disc area for file storage. This allocation takes place when the user initially subscribes to the system and requests disc storage area for his files.

COMMAND LANGUAGE DESCRIPTION (Cont'd)

This storage should be based upon the number of files the user wishes to create and use as his data base. The actual disc storage area reserved is available in "cabinets" that accommodate eight files. A user may request from one to four cabinets. Each file within a cabinet can be defined as a short or long file. A long file is usually reserved for data and may not be used as /CODE, /DO, or FORTRAN TAPEnn files. The user can only access that part of a short file that is normally allocated to source coding (four halfpages). The remaining part of a short file (usually containing the generated object code, etc.) cannot be referenced by a user.

COMMANDS

◆ The commands described below are the only valid system commands. Any other message beginning with a single virgule causes the system to rely 'BAD COMMAND, RETYPE' and expects a subsequent system command. The syntax of each command is illustrated as well as the effect it has on system operation.

$$/\text{ON}\Delta\text{usercode}\ \left[1\text{-}3\ \text{symbols}\right]\ \left[\ \Delta\ \left\{\begin{matrix}\text{OPTIONN}\\ \text{OPTIONP}\\ \text{N}\\ \text{P}\\ \text{OPTION}\end{matrix}\right\}\ \right]$$

This command is given at any time - either upon calling the system, or during a session. The usercode is a name assigned to the user when he becomes a customer of the system. The name is exactly eight characters in length.

The system checks the usercode when this command is received. If the name is valid, the system becomes available to the user; if the name is invalid or unacceptable for some reason, the system breaks the telephone connection.

Following the eight character usercode a user may enter up to a three digit 'initials' field (one to three symbols.) These user initials do not effect the on-line operation of the system. They do, however, appear on the accounting records which makes it possible to keep separate records for several people sharing the same usercode.

If $\left\{\begin{matrix}\text{P}\\ \text{OPTIONP}\end{matrix}\right\}$ is present the user is informed of the current status of the system associated with a specific usercode. When a user first subscribes to the system this status reflects the standards set by the system. Otherwise, the status reflects the options set during a previous time sharing session. If the user is satisfied with the current status he may continue his session. Otherwise, if options are to be changed the user response with $\left\{\begin{matrix}\text{P}\\ \text{BLANK}\end{matrix}\right\}$ after 'READY' is received to obtain additional information concerning the options. The optional word OPTION does not display the current status but provides the additional option information. At this time the user may set global options that are to be associated with all files generated during this and all future time sharing sessions.

COMMANDS (Cont'd)

The information typed out appears as follows:

0-FILES, 1-DEVICE SYMBOLS, 2-MONEY, 3-SECURITY

and the user may now request additional information by typing one of the codes indicated.

This dialogue between the user and the system may be greatly shortened if the user is aware of the status of the system and the procedure required to change it. Entering the OPTIONN or N with the /ON, or N after the 'READY' eliminates the printing associated with OPTION and causes the system to respond with ENTER CHOICE. The user then enters the options of his choice in a single message as the string of symbols he would have sent separately in OPTION.

Notes

◆ 0 - FILES - options that may be set refer to the compiler, file size (long or short), and the Editor (delimiter symbol and verification mode.)

1 - DEVICE SYMBOLS - refers to changing the setting of the backspace character, extension character, or tab stops.

2 - MONEY - allows the user to set the dollar amount (up to the amount remaining in his account) he wishes to spend on this session. Once set, however, it can not be increased for the current session unless another /ON is issued. When the set amount is reached, a /OFF is issued automatically.

3 - SECURITY - allows the user to change the last one-to-three digits of his user code. When a user became a customer, he was assigned an eight character code consisting of five characters and three zeros. These three zeros can be changed by the user.

The entire eight characters are verified for correctness before accepting a usercode. Hence, a user has, in effect, a combination lock to his files. Furthermore, the combination can be changed at any time, but only by someone who knows the old combination. The three-digit suffix does not print out on either the operator's console or in the accounting records. A teletype user can prevent hard copy of his security code at his console by choosing his three-character suffix to contain any of the nonprintable teletype characters.

Once the user has logged on the system, / system commands may be issued at any time.

COMMANDS (Cont'd)

/OPTION [{N} {P}] [Δfilename]

This command provides choices which are to be associated with the current file, and does not affect the global choices previously set. The N suffix is for those who know what to enter, and wish the printing of choices to be suppressed. The P suffix types options currently in effect. At the end of each measage printed, including READY, a blank, N, or P may be entered to begin the process again. Note: refer to the FILES note of the /ON descriotion for additional detail.

/IN [#] [Δfilename]

This command allows the user to enter data into the current file. The system provides a sequence number for each record entered. A linefeed is returned when the system is ready for each new record. If the # symbol appears in the command, the system also displays the number to be supplied for each incoming record.

/CODE [Δfilename]

This command allows the user access to the FORTRAN compiler for the currt file. Once this command has been given input must be wither FORTRAN statements or compiler commands. However, another / - command may be issued at any time to leave the CODE mode.

/EDIT [Δfilename]

This command allows the user access to the EDIT mode for the current file. Once the command is issued, the system is ready to accept input in the EDIT language. However, at any time, the user may issue another / - command to perform another system function.

/PRINT [#] [Δfilename] [Δseq-# [[ΔTO] Δ seq#]]

This command allows the user to print records from the current file. The current file is printed if the user only issued a /PRINT #. If the seq-# option is used the print may range from one record to all of the records specified, starting with the first sequence number and ending with the last. Also, the printed records will have sequence numbers displayed if the # symbol is used with the command.

/RESEQ [Δfilename] [Δseq-#]

This command works the same as /PRINT#, but, in additon, it renumbers starting from the beginning or from the seq-#, if given.

/DO filename [Δseq# [Δyyyy]]

This command can only be used in conjunction with a file that has previously been created. It causes records to be transmitted automatically from the specified file just as though they were being entered from a remote terminal. The DO mode is terminated when a /ENDO is encountered, the DO file exhausts, an error occurs, or the escape key is hit.

COMMANDS (Cont'd)

An additional feature of this command allows the user to change the first four characters of incoming DO file records. The records to be changed must begin with 'XXXX' and these are replaced by any four characters (yyyy) specified in the command. The records in the original file remain unaltered.

/ENDO

This command is used to terminate a DO mode operation.

/LOCK [ΔfilenameΔ] ΔASΔfilename

This command protects a file from change. A copy of the current or first named file is made, given the second name and then locked as the current file. The two filenames may be the same.

/UNLOCKΔfilenameΔASΔfilename

This command makes a copy of the first named file, names the copy with the second name, and unlocks the copy. The copy is established as the current file. The original file is still locked if it had been (unless both file names are identical.)

/DROP filename

This command results in the elimination of the named file from the system, and frees the storage space it had occupied. If the file is locked, it must be unlocked before it can be dropped. After a /DROP command, the last file mentioned is no longer present, so there is no current file.

/CATALOG

This command causes the list of the names of all the files in use by the requesting user to be printed at his console. Those files which have been produced by issuing the command /LOCK are prefixed by the * symbol.

/RENAME [Δfilename] AS filename

This command changes the name of the current file or the first file named to the second name given.

/SAY [Δmessage]

This command merely types back the message given. It is primarily intended to be used within DO files to indicate progress.

/HELP

This command enables the user to obtain a description of the time sharing system. When this command is issued the system responds by printing general information along with a list of valid system commands and their formats. If additional information is required for any of the listed commands, it may be obtained by responding with the two-digit help number printed with the specific command. If the user wishes to terminate the HELP printout he may press the ESC key after the first line as well as during the pause that occurs after every five lines.

COMMANDS (Cont'd)

/COST

This command provides the user with a time and charges report.

/OFF

This command is given when the user wishes to terminate his current session on the system. A time and charges report is produced for the user and the system goes into a wait cycle. This wait cycle lasts approximately 50 seconds before breaking the telephone connection for that console to allow another user to issue a /ON without having to dial for the connection.

Notes:

1. The back arrow (←) key may be used to erase erroneously typed characters. Each time the back arrow key is pressed, one character immediately to the left is erased.

   a. Example:

10	123456 ← ←	INITIAL
10	*1234*	RESULT

2. The ESC key or a single virgule (/) message may be used:

   a. To stop output after every 5 lines (must be pressed during pause)

   b. To stop a process which is strictly internal and is apparently devoid of output.

   c. To delete an erroneous line.

4. EDIT LANGUAGE

ENTERING THE EDIT MODE

◆ The EDIT processing mode is called into use by the user system command /EDIT [Δ filename] . The system is now ready to accept a separate set of input known as the Edit language. These Edit inputs are only recognizable in the Edit mode and are ignored if the user has not previously initiated the Edit processor.

If the file to be edited is current, the user enters the Edit mode by typing /EDIT, otherwise, the user must type /EDIT filename. In either case, the system responds with a line feed and is then ready to accept Edit statements and commands.

STATUS OF EDITOR

Status Delimiter

◆ The Edit processor has at all times a single character marked as the Delimiter symbol. This symbol is used to mark the beginning and end of strings of characters (words, etc) which are to be changed, inserted, found, etc., in the user's records. It is needed in many of the input statements to the Edit system.

Unless changed by the user, the Delimiter will be a colon (:). However, the user may set it - either temporarily during Edit processing or permanently using /OPTION - to any character except a blank. Each time /EDIT is called the Delimiter is set to its permanent value.

Verification

◆ The Edit processor has two verification states - check and nocheck. One verification state is set at all times. When check is set, all changes made to the user's file are printed back to the user. When nocheck is set, the Edit system does the requested processing but does not print the changes back to the user.

During editing, the user has complete control of which Edit operations are to be verified and which are not.

The permanent verification status can be set using /OPTION. Each time /EDIT is called the verification state is set to its permanent value.

Status Line

◆ The Edit system prints out the current delimiter symbol and current verification state whenever desired. Typing in a blank line (1 to 255 blanks) causes the following line to be printed showing the proper symbol and giving a C or N for the verification state.

```
SYM = : VERIFY = C
```

INPUT TO EDIT

Commands and Statements

◆ The Edit system has two types of input -- commands and statements.

Commands are:

HELP

SYMBOL

CHECK,NOCHECK

RESUME,RESUMEC,RESUMEN

LOOP,ENDLOOP

(A Null Record is a special command.)

(A Blank Line is considered a special command.)

The commands exercise general control of the Edit processor. They are not concerned with the details of changes to the user's file.

Statements specify what editing changes are to be made and what records are to be affected. Statements always begin with a list of record numbers. Depending on the type of process being specified, statements may have verbs defining the process, strings of characters, or another list of record numbers.

Input Commands to Edit

HELP

◆ 1. HELP supplies examples and writeups of system usage.

2. Keywords typed after HELP define the writeups desired.
3. The prefix X before keywords marked * supplies examples.
4. Keywords may be strung together with commas.
5. The following keywords are available (need only first 3 letters)

ABNORMAL*	FIND*	SEQ-LIST
ALL	GENERAL	SETPC*
BLANK-LINE	HELP	STATEMENT
CHANGE*	KEYWORD	STRING
CHECK	LOOP*	SUFFIX*
COMMANDS	NOCHECK	SYMBOL
COPY*	NULL RECORD	TEXT
DELETE*	PREFIX*	VERB
ERROR*	PRINT*	WAS*
EXAMPLES	RESUME	

HELP (Cont'd)

6. Examples: (Each will supply a writeup of check and examples of delete.)

   HELP CHECK,XDELETE

   HELP CHE,XDEL

7. Typing only HELP after an error or stop has the same effect as typing a null record; both supply analysis.

SYMBOL

◆ 1. Must be followed immediately by any character except a blank.

2. This character then becomes the current Delimiter symbol.

3. The Delimiter is always reset to its standard value each time /EDIT is entered.

4. Examples:

   SYMBOL: (Sets the Delimiter symbol to :)

   SYMBOL' (Sets Delim to ')

CHECK, NOCHECK

◆ 1. These set the current verify status.

2. The verify status is always reset to its standard value each time /EDIT is entered.

RESUME, RESUMEC, RESUMEN

◆ 1. May be used after the user has stopped the processing.

2. Some error stops also allow resuming.

3. Processing continues where left off.

4. RESUMEC (Resume Check) causes all resumed processing to be verified.

5. RESUMEN (resume nocheck) inhibits all verification.

6. Repeated stops and resumes are allowed.

7. When resume is allowed and some other input is supplied instead, the opportunity to resume is ended. However, a null record may be input at all times.

LOOP, ENDLOOP

◆ 1. LOOP precedes and ENDLOOP follows statements which are to be executed repetitively.

2. LOOP must be followed by a SEQ-RANGE and perhaps a SEQ-NUMBER.

3. For more detail see writeup on LOOP operation.

(NULL RECORD)

- 1. Typing a null record is considered by the Edit system to be a command.
  2. It normally prints the status line.
  3. It prints an analysis of the condition of input after errors or stops.
  4. If in doubt, type in a null record.
  5. Use of a null record input after stops does not affect the opportunity to use resume.

(BLANK LINE)

- 1. A line of from 1 to 255 blanks always prints the status line.

Input Statements to Edit

Statements

- 1. Always begin with a list (called a seq-list) of one or more sequence numbers or ranges.
  2. The general form of the input statements is either:

     seq-list verb string-1 auxiliary verb string-2

     or

     seq-list verb seq-list
  3. Each statement is executed independently of those preceding or following it.
  4. Except during LOOP operation, each statement entered is processed to completion immediately after being entered.
  5. Example: (Will change the word LOG in records 10 and 60 to EXP.)

     10,60 CHANGE :LOG: TO :EXP:

Sequence List and Components - Sequence Number and Range

- *Seq-Number*

  1. All records in the file are identified by their sequence number.
  2. Number must be between 0 and 999999.
  3. Examples:

     94
     09381

Seq-Range

1. Formed by joining two numbers with a dash.
2. Right number must be larger than left.
3. Range is inclusive.
4. Examples:

   007-9
   0-999999

Sequence List and Components - Sequence Number and Range (Cont'd)

Seq-List

1. List is made up of from one to eight sequence numbers and/or ranges joined with commas. No blanks are allowed in a seq-list.

2. Every statement begins with a seq-list.

3. Each element of the beginning seq-list may have a prefix C or N.

4. Such a prefix resets the verify status for the rest (left to right) of that statement only.

5. Examples:

```
5,6,7,2
5                       (List of only one number.)
C1,N2-50,160,C200
```

String

1. A string is a sequence of characters to be found, changed, etc.

2. Strings are specified by putting the Delimiter symbol before and after.

3. If the closing string Delimiter is missing, the string is assumed to end at the end of the record.

4. The Delimiter symbol may occur as a character within a string by showing two symbols for every one desired.

5. Maximum string length is limited only by maximum input length (255).

6. Null strings (those with no characters) are not allowed.

7. Examples: (Delimiter is :)

```
:Z:
:TEN  CHARS:
:SINF                   (Record ends at F)
:ABC::DEF::::GH:        (String is ABC:DEF::GH)
```

Verb

1. The verb describes the type of editing operation desired.

2. One or more blanks must separate the seq-list which begins the statement and the verb.

3. Some verbs have an auxiliary word which follows a string and further defines the processing.

Sequence List and Components - Sequence Number and Range (Cont'd)

4. The verbs are:

Verb	Aux	Must be followed by
PRINT		(nothing or 1 string)
TEXT		(1 string)
CHANGE	TO	(2 strings)
CHANGEALL	TO	(2 strings)
FIND	INSERTP or INSERTS	(2 strings)
FINDALL	INSERTP or INSERTS	(2 strings)
FIND	REPLACEP or REPLACES	(2 strings)
FIND	DROPP or DROPS	(1 string)
DELETE		(nothing or 1 string)
DELETEALL		(1 string)
SUFFIX		(1 string)
PREFIX		(1 string)
SETPC		(1 string must be 1 char only)
COPY		(a seq-list)
WAS		(a seq-list)

5. If a seq-list is followed by only blanks, the verb PRINT is assumed.

6. If a seq-list is followed only by a string, the verb TEXT is assumed.

Errors

Errors in Input

◆ 1. Errors in input commands or statements cause the system to print back the line below with XX set to the error number.

BAD INPUT (#XX)

2. Typing a null record after receiving the above line, prints back an explanation of the cause of the error.

3. Typing a null record after receiving the explanation prints back the input statement or command and a question mark at or before the error location.

Errors During Processing

◆ 1. Execution errors are of three types:

   a. Modified record count exceeds 256 characters.

   b. Record generated by COPY or WAS statements exceeds 256 characters.

   c. A seq-number or range in the right hand seq-list of a COPY or WAS statement is undefined.

2. In type 1 errors, the record in question is left unmodified. Processing stops but can be resumed if there are more records in the list to be processed.

3. Type 2 errors leave all the records involved unchanged. Processing stops and cannot be resumed.

4. Type 3 errors also leave the records unchanged. However, the processing can be resumed if it is desired to skip the particular undefined number or range. During loop operation, type 3 errors are automatically ignored.

5. Typing a null record after all error stops provides more information about the stop condition.

Abnormal Conditions

◆ 1. During the execution of Edit statements (except TEXT, COPY, and WAS statements) two types of things may happen which are considered abnormal:

   a. A seq-number specified is not defined, or a seq-range has no defined records within its range.

   b. A string match is not found in a specified seq-number or at least one match is not found within a seq-range.

2. Processing of the statement is not affected by the abnormal conditions. But, since it may indicate an error in the user's statement, a writeout is made noting the conditions. The write out is made independent of the verify status.

3. Abnormal conditions are not noted during loop operation.

NOTES AND EXAMPLES OF EDIT STATEMENTS

Notes

◆ 1. The records specified by the seq-list beginning each statement are processed in the same order as they occur in the list. Each record is completely processed before the list is examined for the next one.

2. All operations which scan a record for a matching string do so from left to right.

Assumptions for Examples

◆ At the beginning of each set of examples:

1. The user's file contains only three records:

   a. sequence number 1 contains ONE.

   b. sequence number 2 contains TWO.

   c. sequence number 3 contains THREE.

2. The current Delimiter symbol is :.

3. The current verify status is check.

For each example:

1. The user's input is shown beginning at the left margin.

2. *THE SYSTEM RESPONSE IS SHOWN UNDER THE INPUT IN AN ITALIC TYPEFACE.*

3. Notes are given in parenthesis.

Examples

◆ 0 - 999999 (User asks to print all of file.)
1 ONE
2 TWO
3 THREE (System prints the only three records currently in file.)

(User types in one or more blanks to see what the status is.)

SYM = : VERIFY = C (System replies with line showing current Delimiter symbol and verification state.)

0 - 10 PRINTΔ:E: (User wishes to print only records with E.)
1 ONE
3 THREE

Examples *(Cont'd)*

```
8 :ZZ:                  (User creates record 8 as ZZ.)
8 ZZ

3,5 :ABC:               (Records 3 and 5 are defined as ABC.)
3 ABC
5 ABC

2 DELETE                (Remove entire record 2.)
SEQ NUM     2 DELETED.

0-10                    (Print records 0 through 10.)
1 ONE
3 ABC
5 ABC
8 ZZ

2 :TEXT FOR TWO:
2 TEXT FOR TWO

4 COPY 2,1
4 TEXT FOR TWOONE

NOCHECK                 (Set verification to NOCHECK.)
SYM=: VERIFY=N              (System confirms change)

4-5                     (Print statements are not affected by verify status.)
4 TEXT FOR TWOONE
5 ABC

20,21,22,23 :ABC: (Records are formed but not printed back.)
                        (System responds only with a line feed.)

CHECK
SYM=: VERIFY=C              (System confirms check status.)

0-10 DELETE :E: (Delete the leftmost E in each record.)
1 ON
4 TXT FOR TWOONE

4 DELETEALL :0: (Delete all 0's in record 4.)
4 TXT FR TWNE

4 CHANGE :FR: TO :***:
4 TXT *** TWNE
```

Examples *(Cont'd)*

```
1,3,5SUFFIX:.:      (User error - a blank must separate seq-num and
                     verb.)
BAD INPUT(#09)       (System responds.)
                     (User types in null record to request analysis.)
BLANK,COMMA, OR DASH EXPECTED          (System gives analysis.)
                     (User again types in a null record to see input record.)
1,3,5?SUFFIX:.:      (System marks error at or following ? .)

1,3,5 SUFFIX:.:      (User types corrected statement.)
1 ONE.
3 ABC.
5 ABC.
```

COMMAND VERBS

PRINT

Format

◆ seq-list PRINT

seq-list

seq-list PRINT string

Description

◆ 1. Normally all records in the seq-list are printed.

2. If the statement contains a string, only records with a matching string are printed.

3. If only a seq-list is typed in, the print operation is assumed.

4. Print does not change the records in the file.

5. Verify status does not affect print statements.

6. Paper tape can be punched which later can be entered using /CODE or /IN.

Examples

◆ 0 - 999999 Print (Print all of file.)
1 ONE
2 TWO
3 THREE

1,3 - 5 (Assumed print operation.)
1 ONE
3 THREE

N1 (Verify status does not affect print.)
1 ONE

0 - 10 PRINT :T: (Print records with T in them.)
2 TWO
3 THREE

TEXT

Format

◆ seq-list TEXT string

seq-list string (Text operation is assumed.)

Description

◆ 1. Statements consisting of only a seq-list and a string are assumed to be text.

2. The string in a TEXT statement defines a new record.

3. All sequence-numbers given in the seq-list are defined to be the new record.

4. Sequence-ranges in the list have only the new record.

Examples

◆
```
1 TEXT:ABC:
1 ABC

1,6,3 :ZY:
1 ZY
6 ZY
3 ZY

0-999999 :XXX:     (Seq-range causes replacement only.)
1 XXX
2 XXX
3 XXX
6 XXX

4 :WOW             (End of record marks end of string.)
4 WOW

6,N7,8,C9 :SAMP    (Generate but don't check records 7 and 8)
6 SAMP
9 SAMP
```

CHANGE---TO

Format

◆ seq-list CHANGE string-1 TO string-2

Description

◆ Each record specified in the list is scanned for the leftmost occurrence of string-1. If found, it is replaced by string-2.

Examples

◆ 1 CHANGE:E: TO :CE:
1 ONCE

0-10 CHANGE :E: TO :XX:
1 ONCXX
3 THRXXE

3 CHANGE:X:TO:-:
3 THR-X

3,2 CHANGE :-: TO :*:
SEQ NUM 2 UNCHANGED
*3 THR*X*

CHANGEALL---TO

Format

◆ seq-list CHANGEALL string-1 TO string-2

Description

◆ All non-overlapping occurrences of string 1 in the records specified by the seq-list are replaced by string-2. Processing is left to right.

Examples

◆ 3 CHANGEALL :E: TO :*R:
*3 THR*R*R*

3 CHANGEALL :R*R: TO :-:
*3 TH-*R*

DELETE

Format

◆ seq-list DELETE

seq-list DELETE string

Description

◆ 1. If a string does not follow the verb, all records specified by the seq-list are completely deleted from the file.

2. When a string is given, each record in the list is scanned for the leftmost occurrence of a matching string. If found, the string is deleted from the record. If the record consists only of the string, then the entire record is deleted, and the seq-number becomes undefined.

Examples

◆ 1 DELETE
SEQ NUM 1 DELETED

2-10 DELETE :R:
3 THEE

3 DELETE :THEE: (String given here is entire record)
SEQ NUM 3 DELETED

N0-999999 DELETE (Delete entire file contents--don't check)

DELETEALL

Format

◆ seq-list DELETEALL string

Description

◆ All non-overlapping occurrences of the string in each record specified by the list are deleted from the record. Processing is from left to right.

Examples

◆ 3 DELETEALL:E:
3 THR

5 :ABABABA (Form record to show example below)
5 ABABABA

5 DELETEALL :ABA:
5 B (Non-overlapping strings were deleted)

FIND---INSERT(P)(S)

Format

◆ seq-list FIND string-1 INSERTP string-2

seq-list FIND string-1 INSERTS string-2

Description

◆ 1. The leftmost occurrence of String-1 is found in each record specified by the list.

2. INSERTP (Insert prefix) places String-2 directly in front of the matching string.

3. INSERTS (Insert suffix) places String-2 directly after the matching string.

4. All characters previously in the record remain.

Examples

◆ 1-3 FIND :0: INSERTP :*:
*1 *ONE*
*2 TW*O*

1,3 FIND:E:INSERTS:-:
*1 *ONE -*
3 THRE-E

FINDALL---INSERT(P)(S)

Format

◆ seq-list FINDALL string-1 INSERTP string-2

seq-list FINDALL string-1 INSERTS string-2

Description

◆ Works exactly as FIND---INSERT(P) (S) except that all non-overlapping occurrences of string-1 are treated rather than just the leftmost one.

Examples

◆ 3 FINDALL :E: INSERTP :*:
*3 THR*E*E*

4 :ABABABA: (Formed to show example below.)
4 ABABABA

4 FINDALL :ABA: INSERTS :*:
*4 ABA*BABA**

FIND---REPLACE(P)(S)

Format

◆ seq-list FIND string-1 REPLACEP string-2

seq-list FIND string-1 REPLACES string-2

Description

◆ 1. As each record is processed, the left most occurrence of string-1 is found.

2. Everything before string-1 is the prefix (may be null.) Everything after string-1 is the suffix (may be null.)

3. REPLACP (replace prefix) replaces the prefix with string-2.

4. REPLACES (replace suffix) replaces the suffix with string-2.

Examples

◆ 1 FIND :N: REPLACES :.:
1 ON.

3 FIND :E: REPLACEP :-:
3 -EE

FIND---DROP(P)(S)

Format

◆ seq-list FIND string-1 DROPP

seq-list FIND string-1 DROPS

Description

◆ 1. As each record is processed, the leftmost occurrence of string-1 is found.

2. Everything before string-1 is the prefix (may be null.) Everything after string-1 is the suffix (may be null.)

3. DROPP (drop prefix) deletes the prefix. The changed record begins with the matching string.

4. DROPS (drop suffix) deletes the suffix. The changed record ends with matching string.

Examples

◆ 1 FIND :E: DROPP
1 E

3 FIND :R: DROPS
3 THR

PREFIX and SUFFIX

Format

◆ seq-list PREFIX string

seq-list SUFFIX string

Description

◆ The string given is placed at the beginning (prefix) or end (suffix) of the specified records.

Examples

◆ 1,3 SUFFIX :.:

```
1 ONE.
3 THREE.
```

1 PREFIX :REC-:

```
1 REC-ONE.
```

SETPC

Format

◆ seq-list SETPC string

Description

◆ 1. The string must be exactly one character.

2. Each record in the file has associated with it a character called the print control. It determines the line advances made before printing the record.

3. All print control characters are normally set to give one line advance before printing.

4. If the string in a SETPC statement is a one-digit number, the number of lines advanced before printing are set to 0 through 9 as specified.

5. Other characters give from 0 to 14 line advances or a page skip. In particular, the character P causes the changed records to do a page skip.

6. Edit operations (except SETPC) do not change the print control character. However, new or replacement records generated by TEXT, COPY, and WAS operations have a single line advance character.

Examples

◆ 1 SETPC :2: (Skip two lines before printing.)

```
1 ONE
```

1 SETPC :4: (Skip four lines.)

```
1 ONE
```

N2,3 SETPC :P: (Records 2 and 3 cause page skip when printed.)

COPY

Format

- seq-list COPY seq-list

Description

- 1. Records specified by the right-hand seq-list are concatenated in the order listed (i.e. ,strung together) to form one new record.

  2. The new record is placed in each of the sequence-numbers given in the left-hand seq-list.

  3. Sequence-ranges in the left-hand list have all previously defined records replaced by the new record.

Examples

- 5,6 COPY 2,1
  5 TWOONE
  6 TWOONE

  1 COPY 1-3
  1 ONETWOTHREE

  4-999999 COPY 2 (Range causes replacement only.)
  5 TWO
  6 TWO

WAS

Format

- seq-list WAS seq-list

Description

- 1. Records specified by the right-hand seq-list are concatenated in the order listed (i.e., strung together) to form one new record.

  2. When the new record is formed, all records in the right-hand seq-list are deleted.

  3. All sequence-numbers given in the left-hand seq-list are defined to be the new record.

  4. Sequence ranges in the left hand list have all previously defined records replaced by the new record. (But see last example.)

Examples

- 5 WAS 1 (Delete record 1.)
  5 ONE

  1 WAS 5,2 (Delete records 5 and 2.)
  1 ONETWO

  0 - 10 WAS 2 - 99 (Delete 2 - 99 before replacing 0 - 10.)
  1 THREE

LOOP OPERATIONS

Format

◆ LOOP seq-range, seq-number

LOOP seq-range

Description

◆ 1. A loop operation is similar to a DO loop in FORTRAN.

2. One begins a LOOP operation by typing the command LOOP followed either by a sequence range or a seq-range and seq-number. For example:

```
LOOP 0-10,1
LOOP 10-200,5
LOOP 0-900
```

The range defines the limits of an index variable which is set and incremented as the loop is executed.

3. Statements and commands are typed-in one-by-one after the LOOP command. The system analyzes each for errors. If there are none, the system responds with a line feed. The statements are not acted upon but are stored in the system for later execution.

4. When all of the statements to be used have been entered the command ENDLOOP is typed-in. It signals the system to begin processing of the loop statements.

5. The system sets the index variable to its initial value (described later on) and begins executing the stored statements in the loop.

6. When all the statements have been executed, the index variable is incremented and tested to see if it is outside the range that was given in the LOOP command.

7. If the index is within the range, the stored statements in the loop are again executed and the incrementing process continues.

8. When the index exceeds the upper end of the range, the system will have completed the LOOP operation and will expect more input statements or commands from the user.

9. Before any more detail, consider the following example:

```
LOOP 1-4,1
1 PREFIX :*:
ENDLOOP
```

Description *(Cont'd)*

The user typed in the above three input lines. After lines one and two the system responded with a line feed to show that the lines had been accepted correctly. When line three was put in, the system began to execute the loop. The loop (in this case only 1 statement) was executed four times. If we assume that sequence number 1 contained the three letters, ONE, before the processing, the system output from the LOOP operation would have been--

```
1 *ONE
1 **ONE
1 ***ONE
1 ****ONE
```

10. Consider the following two methods in which the index variable is set initially and then incremented.

    a. When the LOOP command is followed by both a seq-range and a seq-number, as for example -

    LOOP 1-10,3

    The index value is initially set to the low end of the range.(In the example above, it would be set to 1.) At the end of each pass through the loop, the index is incremented by the number given. (In the example, the increment would be 3.) The incremented value of the index is compared to the high end of the range (10 in the example), and processing stops if the index exceeds the limit. Thus in the example, the index would take on the values 1,4,7 and 10. Four passes would be made through the loop.

    b. If the LOOP command is followed by only a seq-range, as, for example:

    LOOP 10-900

    The index variable takes on values which depend on the sequence numbers which are defined in the user's file at the time.

    In particular, the user's file is examined to find the lowest defined seq-number which is within the given range. The index is initially set to this value. At the end of each pass through the loop, the index value is advanced to the next larger seq-number defined at that time. If this value is still within the range given by the LOOP command, another pass is made through the statements of the loop.

Index Use in Sequence Numbers

◆ 1. Statements within a loop (that is between the commands LOOP and ENDLOOP) may use the index variable to form seq-numbers which change in value at each loop pass.

2. The index variable is represented by the character @.

3. A constant (one to six digits between 0 and 999998) may be added to the index to form a floating seq-number. For example:

```
@+16
@
@+995380
```

4. The sum of the constant added to the index and the maximum range of the index are checked to see that it cannot exceed 999999.

5. Floating seq-numbers may be used in a seq-range, but both ends must be floating. For example:

```
@-@+15
@+5-@+7
```

6. Floating numbers and ranges may be used just as any others in forming seq-lists for statements within the loop. In particular they may be prefixed with C or N just as regular numbers and ranges.

7. Example using index variable:

```
LOOP 0-200,2
N@ :ABC:
N@+ :DEF:
ENDLOOP
```

Records 0,2,4,6,....,200 are formed , each consisting of three letters ABC. Records 1,3,5,....,201 are formed consisting of the three letters DEF. None of the records are written back to the user.

Special Rules for Processing Statements in Loop Operation

◆ 1. All abnormal conditions are ignored during Loop operations.

2. Undefined seq-numbers and ranges in the right-hand seq-list of statements (COPY and WAS statements) are simply ignored unless the entire right-hand list is undefined. If all elements of the list are undefined, the COPY or WAS process are performed by deleting (that is, making undefined) all elements of the left-hand seq-list.

3. Rules 10 a and 10 b allow the user to write complex search loops without having error stops and undesired writeouts.

Examples

◆ 1. Object is to put 200 records into a file. Each record is ABCDE and the record numbers are to be 10,20,30,....,2000. The records are not to be printed back to the user as they are formed.

```
LOOP 10-2000,10
NOCHECK
SYM=: VERIFY=N          (System response to NOCHECK command.)
@ :ABCDE:
ENDLOOP
```

2. Object is to go through all the records of a file (file may have many defined records) and to delete the leftmost two asterisks (*) from each record. If only one asterisk is found, delete it. Verification of the changes is not desired.

```
LOOP 0-999999
N@,@ DELETE :*:
ENDLOOP
```

3. Assume that a file contains records with seq-numbers from 10 to 850. One wishes to rearrange the records in the file so that those originally numbered 150 through 320 are placed at the end of the changed file. The changes, are not to be verified.

```
LOOP 150-320
N@+1000 WAS @
ENDLOOP
```

4. Object is to go through a file of may records and print only those records ending with a question mark (?). Assume that some character can be found (say #) that is known not to occur in the records.

```
LOOP 0-999999
N@ SUFFIX :#:
@ PRINT :?#:
N@ DELETE :#:
ENDLOOP
```

The records ending with ? are printed out. In the print-out, each record ends with ? #. If instead of #, a non-printable character had been used, this problem would not occur.

5. Object is to delete the second asterisk (*) from the left in all records numbered 0 through 1000. The results are not to be verified. Assume that some character can be found (say #) that is known not to occur in the records.

```
LOOP 0-1000
N@ CHANGE :*: TO :#:
N@ DELETE :*:
N@ CHANGE :#: TO :*:
ENDLOOP
```

5. INTERACTIVE FORTRAN COMPILER

GENERAL

◆ The Basic Time Sharing System Interactive FORTRAN compiler provides a means for expressing and executing computational precedures on a computer-controlled remote terminal.

It contains the algorithmic portion of the language plus additional features oriented towards more efficient systems utilization in a time-sharing environment.

The Interactive FORTRAN language is conversationally oriented; e.g., line-for-line diagnostics are produced and the option of immediate execution of a statement is available. In addition, special debugging commands are available to assist the user in implementing his program.

USING THE INTERACTIVE COMPILER

◆ In order to call the interactive FORTRAN compiler, the following statement is typed anytime the system is in the ready-to-receive state:

/CODE Δ [filename]

The system responds with the next available sequence number (number 10 when the file is new).

When using either FORTRAN code or the special features which are accessed through the use of compiler commands, column 1 starts immediately following the sequence number. Columns 1 through 5 are used for FORTRAN line numbers; column 6 must contain either a space or an *; columns 7 through 255 are used for both FORTRAN Code and Compiler Commands; all spaces are ignored. In order to continue on subsequent lines, LINE FEED and RETURN are typed in that order. A statement is terminated when RETURN is typed.

COMPILER COMMANDS

◆ The source text is edited and compiled on a line-for-line basis in every case, but may be saved and/or executed depending on the compiler command(s) in effect.

EXECUTE Δ [{ ON }]
[{ OFF }]

COMPILER COMMANDS
(Cont'd)

When the execute switch is set 'ON' each statement is executed after it is compiled. The result of the execution is displayed to the user in the form of trace and flow statements (see TRACE and FLOW debugging aids.) Setting this switch 'ON' is useful in testing the initial values through program loops and in program debugging.

Notes:

1. The normal state of the Execute switch is 'OFF'.

2. During compilation and execution of a subroutine only the following FORTRAN statements are permitted if the Execute switch is 'ON':

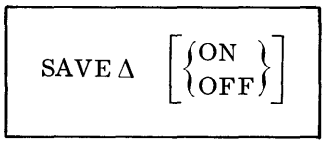
   DUMP (no list)

   PDUMP (no list)

   RETURN

   TRACE ON/OFF (no list)

   FLOW ON/OFF



When the Save switch is set 'OFF' all following statements are compiled (and executed if the Execute switch is 'ON') but the source statement and the compiled code are not saved in the user's file.

Notes:

1. The normal state of the save switch is 'ON'.

2. Variables defined in statements processed with the save switch 'OFF' retain their computed values and may be referenced by later statements.

3. If the Save switch is 'OFF' no statement with a FORTRAN line number can be processed.

4. If the input to the compiler comes from a user's file, then the Save switch has no effect on the source statement and the compiled code is not saved.

COMPILER COMMANDS (Cont'd)

```
CALCULATOR
```

This command has the combined effect of both an EXECUTE ON and a SAVE OFF command. In order to reset these switches, the appropriate EXECUTE and SAVE commands must be given.

```
*
```

An asterisk preceding any statement causes the Execute switch to be set 'ON' and the Save switch 'OFF' for the processing of only that statement. After this statement has been processed these switches are returned to their appropriate values.

Notes:

1. The technique of preceding statements with an * should be used to keep unwanted compiler commands and debugging statements out of the user's file.
2. In Boolean statements the * should appear in column 6.

The compiler commands affecting control of program execution are START, HALT, and RESET.

```
START [ blank
        Δline number
        Δ*sequence number ]
```

Execution of the current program is either started or continued at the specified line or sequence number. If no control number is given; that is, (blank), the program starts or resumes execution after the last executed statement. When running under the inplied mode; that is, EXECUTE OFF, SAVE ON, file execution can be started by pressing the RETURN key. This is the same as if *START had been typed by the user.

```
HALT [ blank
       Δline number
       Δ*sequence number ]
```

This command sets a program stop at the specified line or sequence number, or resets all previous HALT commands if no control number is given.

COMPILER COMMANDS (Cont'd)

```
RESET [ blank
        Δ line number
        Δ *sequence number ]
```

This command resets the compiler to accept input for the specified line or sequence number. When this command is given the compiler accepts a statement which replaces the statement specified by the line or sequence number. After accepting the new statement, the current file is recompiled. A RESET command with no parameter (blank) causes recompilation. It should be noted that this command inserts a line as well as replaces existing lines. Insertion of a line occurs if the "*sequence number" does not currently exist in the file.

Note:

The user is cautioned that the compiler commands used are part of the current interactive file, if they are not preceded by an '*'.

Examples:

Column 7

*START*90; start execution at sequence number 90.

*START 100; start execution at line number 100.

*START; start execution at last non-executed line.

*HALT*115; program to stop at sequence number 115.

*RESET; recompile program.

*RESET*100; reset line at sequence number 100.

*RESET*25; insert line between 20 and 30.

DEBUGGING AIDS

◆ Three commands are available to assist the user in debugging his Interactive FORTRAN program. These are DUMP and PDUMP, TRACE, and FLOW.

```
PDUMP Δ list

DUMPΔlist
```

Both these commands cause a printout of all the listed variables occurring in that program (or subroutine.) Only those variables on list are printed. PDUMP continues execution of the program while DUMP terminates program execution allowing user action. If there is no list then all variables are printed.

```
TRACEΔ [ON ] list
       [OFF]
```

This command causes any variable in the list to be printed out when ever it is altered during execution. If there is no list, all variables are traced.

DEBUGGING AIDS (Cont'd)

FLOWΔ [ON / OFF]

The FLOW command traces all transfers of control and causes printouts indicating the sequence numbers involved in the programs' branching logic.

PROGRAM PREPARATION

◆ The source program is entered into the system from a remote terminal on a line for line basis. Continuation lines are not permitted. A source statement consists of the normal FORTRAN form; positions 1-5 for line line-number, position 6 must be blank or an *, positions 7-255 for statement syntax. A letter C in column 1 designates a comment line. An extended feature of the Interactive Compiler permits an insertion of a semicolon (;) after any FORTRAN statement and means the text following in the line is comment.

For example:

```
X=Y**2-1;EQUATE X TO RADICAL
PRINT 10,I,Y; PRINT PARAMETERS
```

The Interactive FORTRAN character set consists of numerics 0-9, alphabetics A-Z and the following special characters:

Character	Name
=	equal
+	plus
-	minus
*	asterisk
/	slash
.	decimal point
,	comma
(	left parenthesis
)	right parenthesis
$	dollar sign
'	apostrophe

Any other character(s) of the complete 256 character EBCDIC set may be used only within a literal constant.

LANGUAGE ELEMENTS

◆ The elements of the Interactive FORTRAN language statements are constants, variables, and arrays of variables.

Constants

◆ Constants may be numeric or literal by class, and integer or real by type. An integer or real constant may be negative, zero, or positive, and must be of the allowable range.

Integer Constant Range

0 to 536870911: i.e., $(2^{29} - 1)$

Real Constant Range

16^{-65} to 16^{63} (i.e., approximately 10^{75})

Variables

◆ A variable is a symbolic representation of a quantity that may assume different values. A variable may be real, integer, or complex. A variable name consists of from one-to-six alphanumeric characters (0-9, A-Z), the first of which must be alphabetic. Additional characters used in a name will be ignored.

An integer variable has a standard length of four bytes, which is sufficient to contain the maximum integer value. The internal value of an Interactive FORTRAN integer is four times that of an equivalent machine integer; hence, the maximum value of $2^{29}-1$.* A real variable has a standard length of four bytes assigned to its value and an optional double precision length of eight bytes.

A complex variable consists of two 8-byte double precision parts. No optional single precision is available.

The predefined specification of FORTRAN applies; that is, if the first character of the variable name is I,J,K,L,M, or N, the variable is an integer. The explicit specification statements of REAL and INTEGER can be used to designate the type of specific variable names as other than the implicit standard. The specification statement, DOUBLE PRECISION, is used to assign the optional maximum real variable length of 8. D.P. is an acceptable abbreviation for DOUBLE PRECISION.

ARRAYS

◆ Arrays of variables may be defined through the use of the COMMON statement or an explicit Type statement only (no other specification statement such as DIMENSION or EQUIVALENCE is permitted.) The number of dimensions is unlimited for any array.

An array subscript may be any constant, simple variable, variable, or integer expression. In the case of a multi-dimensioned array, no limit is placed on any one of the subscripts with regard to their format. Whatever subscript form is used, however, its evaluated result should be greater than zero and less than or equal to the limit specified for it.

*The two low-order bits are not part of the Interactive FORTRAN integer representations. They are, however, significant in Boolean expressions.

INTERACTIVE FORTRAN LANGUAGE

◆ The following sections present the syntax of each of the permissible FORTRAN statements.

Arithmetic Expressions

◆ Arithmetic expressions range in complexity from single constants or variables to expressions containing two or more constants and/or variables connected through arithmetic operators. The arithmetic operators are:

Arithmetic Operator	Definition
**	Exponentiation
*	Multiplication
/	Division
+	Addition
-	Subtraction

Standard FORTRAN rules for constructing arithmetic expressions apply.

Arithmetic expressions may be mixed in mode; that is, they may contain integer, real, and complex constants and/or variables.

Arithmetic Assignment Statement

◆ The standard form for assignment is V = e where V is any variable and e is any arithmetic expression. The equal sign specifies replacement rather than equivalence.

The valid assignment statements are as follows:

Assignment Type*	Mode of Computation*	Assignment Manipulation
Integer = Integer	Integer	None
Integer = Real	Real	Truncated to Integer
Integer = D. P.	D. P.	Truncated to Integer
Integer = Complex	Complex	Truncate and Assign Real Part only
Real = Integer	Integer	Converted to Real
Real = Real	Real	None
Real = D. P.	D. P.	Truncated to Real
Real = Complex	(Not allowed)	—
D. P. = Integer	Integer	Converted to D. P.
D. P. = Real	D. P.	Converted to D. P.
D. P. = D. P.	D. P.	None
D. P. = Complex	Complex	Only Real part assigned
Complex = Integer/ Real/D.P./Complex	Complex	None

In any case of mixed mode, the highest mode dictates the mode of computation.

*When Real is used, it represents standard length (full word) real numbers; D.P. represents double precision (double word) real numbers.

COMPLEX VARIABLE ASSIGNMENT

Assignment of constants to complex variables deviates somewhat from current standards. The following examples illustrate how values are assigned:

(Given the general complex form (a + bi))

```
COMPLEX C
C = 1,1          ;a = 1, b = 1
C = (1,1)        ;a = 1, b = 1
C = -1,1         ;a = -1, b = -1
C = -(1,1)       ;a = -1, b = -1
C = (1,0) + (0,1);a = 1, b = 1
C = -(1,0)+(0,1);a=-1,b=1
C = (1,0) - (0,1);a = 1, b = -1
C = 1, -1        ;INVALID
C = (1, -1)      ;INVALID
```

The printing of both the real and imaginary parts of a complex number cannot be accomplished by using the PRINT statement. The following examples illustrate how the '*' and the PDUMP statement can be used to print complex numbers:

```
C    N O T E :
C     THE FOLLOWING EXAMPLES DO NOT DIFFERENTIATE
C     BETWEEN SYSTEM AND USER RESPONSE SINCE THEY
C     ARE A REPRODUCTION OF AN ACTUAL TERMINAL SESSION


/CODE CMPLX
      50        COMPLEX X,Y,Z
      60        *X=1,1
X = ( .100000000000000000E 01, .100000000000000000E 01)
      60        *X=(1,1)
X = ( .100000000000000000E 01, .100000000000000000E 01)
      60        *X=(1,0)+(01←,1)
X = ( .100000000000000000E 01, .100000000000000000E 01)
      60        *X=-1,1
X = (-.100000000000000000E 01,-.100000000000000000E 01)
      60        *X=-(1,1)
X = (-.100000000000000000E 01,-.100000000000000000E 01)
      60        *X=8←(-1,1)
X = (-.100000000000000000E 01,-.100000000000000000E 01)
      60        *X=-(1,0)+(0,1)
X = (-.100000000000000000E 01, .100000000000000000E 01)
      60        *X=8←(-1,0)+(0,1)
```

COMPLEX VARIABLE ASSIGNMENT (Cont'd)

```
X = (-.10000000000000000E Ø1, .10000000000000000E Ø1)
     6Ø         *X=1,-1
     6Ø ERROR    X=1,-← 1
     6Ø         *X=(1,-1)
     6Ø ERROR   X=(1,-← 1)
     6Ø         *X=1,-(+1)
     6Ø ERROR    X=1,-← (+1)
     6Ø         *X=(1,Ø)-(Ø,1)
X = ( .10000000000000000E Ø1,-.10000000000000000E Ø1)
     6Ø         *X=1,Ø-Ø,1
X = ( .10000000000000000E Ø1,-.10000000000000000E Ø1)
     6Ø 1Ø      FORMAT(1X2E16.8)
     7Ø         X=(1,Ø)-(Ø,1)
     8Ø         PRINT 1Ø,X
     9Ø         *START*7Ø
  .10000000000000000E Ø1
STOP EXECUTED AFTER *8Ø
     9Ø         *P
     9Ø         PDUMP X
    1ØØ         *START*7Ø
  .10000000000000000E Ø1
X = ( .10000000000000000E Ø1,-.10000000000000000E Ø1)
STOP EXECUTED AFTER *9Ø
    1ØØ         *X←Y=MAG(X)
Y = ( .14142135623730941E Ø1, .00000000000000000E ØØ)
    1ØØ         *PDUMP X,Y
X = ( .10000000000000000E Ø1,-.10000000000000000E Ø1)
Y = ( .14142135623730941E Ø1, .00000000000000000E ØØ)
    1ØØ
STOP EXECUTED AFTER *9Ø
    1ØØ
```

Boolean Expressions

◆ An extended language feature of the Interactive FORTRAN is the ability to link arithmetic constants and variables with boolean operators. This feature is invoked by inserting the letter B in position 1 of the FORTRAN statement. The boolean operators are:

Boolean Operator	Definition
*	AND
+	OR
-	NOT

Boolean Expressions *(Cont'd)*

Any other normal arithmetic operator when using the B option produces error printouts.

Only integer and real variables of standard length are permitted in a boolean expression. Constants permitted are 1 or .TRUE. for true, and 0 (zero) or .FALSE. for false. The machine representation for true is a word of binary 1's and for false, a word of binary 0's.

Note:

The constants and variables in a boolean expression are considered as TRUE if other than zero.

Boolean IF Statement

◆ The form for the Boolean IF statement is:

Column 1	Column 7
B	IF (e) n_1, n_2, n_3

where:e is a boolean expression, n_1, n_2, n_3 are statement numbers. Control goes to statement n_1 if the value of e is negative (TRUE), n_2 if it is zero (FALSE), and n_3 if it is positive (TRUE).

Control Statements

◆ Normally, statements are executed sequentially. This section discusses the statements that may be used to alter and control the normal sequence of execution of statements in the program.

GO TO STATEMENT

◆ The unconditional GO TO n statement transfers control to statement n which is the next statement executed. Statements are executed consecutively from statement n until another control statement changes the sequence.

ARITHMETIC IF STATEMENT

◆ The IF statement form is:

IF (e) n_1, n_2, n_3

where e is an integer, real, or double precision arithmetic expression; n_1, n_2, n_3, are statement numbers of executable statements. Control goes to statement n_1 if the value of e is negative, n_2 if it is zero, and n_3 if it is positive.

LOGICAL IF STATEMENT

◆ The form of the LOGICAL IF statement is:

IF (e·RR·e) S

where e is an integer, real, or double precision arithmetic expression; S is a FORTRAN statement; and .RR. is one of the relational operators in the following list:

·RR·	Definition
.LT.	Less than
.LE.	Less than or equal to
.EQ.	Equal
.GE.	Greater than or equal to
.GT.	Greater than
.NE.	Not equal

The following are valid logical IF statement:

```
IF (X.GE.Y) GO TO 99
IF (Z*Z.LE.ENDPT) CALL NODE (DERX,Z,ETC)
```

DO STATEMENT

◆ The form of the DO statement is:

DO n i = m_1, m_2, m_3

where n is the statement number of subsequent executable statement, i is a non-subscribed integer variable called the index, and m_1, m_2, m_3 are integer constants, integer variables, or integer expressions. If m_3 is not present, it is assumed to be 1.

Nested DO statements are permitted to any depth. Transfer out of any DO loop is permissible. Transfer into a DO loop is permitted only if it is the innermost nest, and a previous transfer was made out of the same innermost range; it is assumed that none of the indexing parameters are changed in the interim outside the range of the DO.

A DO loop may not end with a GO TO statement, an Arithmetic IF, or another DO statement.

CONTINUE STATEMENT

◆ The CONTINUE statement may be placed anywhere in the source program without affecting the sequence of execution. It is normally used as the last statement in the range of a DO to avoid ending the DO loop with a GO TO, Arithmetic IF, or another DO statement.

RETURN STATEMENT

◆ The RETURN statement must be used one or more times in a SUBROUTINE and returns control to the next statement following the CALL in the calling program. The use of the RETURN statement in the main program is an error which is detected during execution.

END STATEMENT

◆ The END statement may be used to define the end of a source program and must be used to define the end of a subroutine. The END statement may not be used until all DO loops have been terminated.

STOP STATEMENT

◆ The STOP statement terminates the execution of the object program and returns control to the terminal with the printout:

STOP EXECUTED AT_____

The user can continue execution at the next executable statement by issuing an *START (see page 5-3).

CALL STATEMENT

◆ The CALL statement is of the form:

CALL name $(P_1,P_2,\ldots P_n)$

where name is the name of a SUBROUTINE subprogram and $P_1, P_2,\ldots,P_n$ are the parameters that are being supplied to the SUBROUTINE subprogram.

The parameters in a CALL statement may be subscripted or non-subscripted variables, subroutine names, or array names. The parameters in a CALL statement must agree in number, order, and type with the corresponding parameters in the SUBROUTINE subprogram.

Note:

Constants or evaluated expressions cannot be passed as parameters in a CALL statement.

SUBROUTINE STATEMENT

◆ The form of the SUBROUTINE statement is:

$$\begin{bmatrix}\text{SUBROUTINE}\\ \text{SUBR.}\end{bmatrix} \text{ name } (P_1,P_2,\ldots,P_n)$$

where name is the subroutine name and $(P_1,P_2,\ldots,P_n)$ are parameters as described in the discussion of the CALL statement. The parameters may be considered as dummy variable names that are replaced at the time of execution by the actual arguments supplied in the CALL statement.

Since the SUBROUTINE is a separate subprogram, the variable names and statement numbers within it do not relate to any other program, with the obvious exception of COMMON variables positionally renamed in a subprogram COMMON statement. The subroutine name is not a variable of the subprogram.

Input/Output Statements

◆ The input/output statements enable a user to enter data as specified by a list from the remote terminal, to transfer computed results from the current file to the remote terminals, or to read (or write) data to on-line data files associated with the current interactive file.

Data is transferred under the control of a FORMAT statement referred to in an I/O statement. The FORMAT statement provides the capabilities of subdividing data into records and declaring the form in which the data is to be transmitted.

READ STATEMENT

◆ The form of the READ statement is:

READ f, list

or

READ (Tapeno,f) list

where f is the statement number of the FORMAT statement describing the data being read; list is a series of variable names, separated by commas, which may be subscripted and specifies the number of items to be read and the symbolic locations in storage into which the data is placed (implied DO's are not allowed); Tapeno is an integer expression whose value (1-99) denotes one of the user on-line files containing the prestored data required by the current interactive file. If the files are created in /EDIT mode they must be named TAPEnn, where nn is between 1 and 99 inclusive. READs and WRITEs to Tape 00 are diverted to the teletype.

The (READ f, list) form of the READ statement does not imply card reader input as does traditional Basic FORTRAN. The user has the capability of inputting data from the remote terminal during the execution phases of his current interactive file through the use of the list items. When a (READ f, list) statement is executed the first variable name of the list is typed back to the remote terminal requesting an assignment for each variable in the list.

For example:

```
Source - 10     4  FORMAT   (2F5:2)
         20        READ  4,X,Y
```

At Execution Time:

```
X = 42.31
Y = 8.20
     or
X = 42.31,8.20
```

PRINT STATEMENT

◆ The form of the PRINT statement is:

PRINT f, list

where f is the statement number of a FORMAT statement, and list is a series of variable or array names, separated by commas.

PRINT STATEMENT (Cont'd)

The PRINT statement transfers results from current file execution to the remote terminal under control of the specified format. The PRINT statement does not reference any other printing device, only the remote terminal.

WRITE STATEMENT

◆ The form of the WRITE statement is:

WRITE (Tapeno,f) list

where f is the statement number of the FORMAT statement describing the data being written: list is a series of variable names, separated by commas; Tapeno is an integer expression whose value (01-99) denotes one of the user online files that is to accept data written from a FORTRAN file (see discussion of REWIND for creating data files). A Tapeno value of 00 will result in the selection of the terminal unit as the output unit.

REWIND STATEMENT

◆ The form of the REWIND statement is:

REWIND tapeno

where Tapeno is an integer expression whose value (01-99) denotes a current data file or one that is being created.

The REWIND statement positions a file to sequence number 0 (zero). If the Tapeno specified is a new file number, a new file is created with a record of sequence number zero containing CT SENTINEL. If the Tapeno specified is an established file number but has no record zero, a sequence zero is created containing CT SENTINEL. Thereafter, the next WRITE will start with sequence number 10; the next READ will start at the next sequence number whatever it is.

BACKSPACE STATEMENT

◆ The form of the BACKSPACE statement is:

BACKSPACE Tapeno

where Tapeno is an integer expression whose value (01-99) denotes a current data file.

This statement positions the file to the previous record. A Tapeno of 0 (zero) is illegal since it references the terminal device.

FORMAT STATEMENT

◆ The FORMAT statement is used in conjunction with the READ, WRITE, and PRINT statements to specify the desired form of the data to be transferred. The form of the data is varied by the different format codes. FORMAT statements are nonexecutable and may be inserted anywhere in the current interactive file. A FORMAT statement may be used to define a FORTRAN record through the use of slashes and parenthesis.* In BTSS FORTRAN a FORMAT statement must precede its use in an I/O statement.

The format codes available in interactive FORTRAN are:

Numeric (Real)

aEw·d
aFw·d

where a is optional and is an unsigned integer constant used to denote the number of times the same format code is repetitively referenced; w is an unsigned integer constant specifying the total field length of the data; and d is an unsigned integer constant specifying the number of decimal places.

Notes:

1. d is always assumed zero; the decimal point or E notation denotes the actual decimal specification.

2. The F format is interpreted as E format.

Numeric (Integer)

aIw

where a is an optional repeat specification and w is an unsigned integer constant specifying the total field length of the data.

An extended feature of the I format is the specification aI0; that is, a field length specification of zero. This definition means that at run time the data itself will specify the characters required.

For Example:

If the machine representation of I,J,K were 30, 3000, and 300000 respectively, the following would occur:

```
30   10  FORMAT   (1H_, 5X,I0/5X,I0/5X,I0)
40       *PRINT    10,I,J,K
 30
 300
 30000
```

* For a more detailed discussion, see pages 8-21 through 8-23 of the TOS FORTRAN IV Reference Manual, No. 70-00-604.

FORMAT STATEMENT (Cont'd)

Alphanumeric and Hexadecimal

a A w
a O w

where a is as above, and w specifies the number of characters of data; A being alphanumeric; O (letter) being hexadecimal.

Spacing

w X

where w specifies the number of spaces to be inserted on output or the number of characters to be skipped on input.

Literal

either w H or quotes

where w specifies the number of characters following H.

The following features of the Interactive FORTRAN formatter should be noted:

Input - 1) When using E, F, or O format, the presence of a comma within the field overrides the specified length. The next field is assumed to start after the comma. 2) Null fields (,,) are not allowed. 3) the d portion for real numbers is ignored. Spaces are ignored except for "counting" and are not assumed to be zero. 4) E and F are both interpreted as E and allow anything that can be compiled within the length constraint. 5) I format, other than I0 assumes an exact count if not controlled correctly by a comma. The following case should be noted:

```
10  FORMAT  (1H_5X,3I4)
    READ  10, I, J, K
    *START
/=  403, 3636, 27
```

The first group of four ends with a comma (403,). The next group of four (3636) is accepted but the last group (,27) is unacceptable. To ensure no problems, the user should make the I specification either zero or 1 larger than the largest integer.

E and F formats have the following standards regardless of FORMAT specification:

Output - 1) single precision = E14.7

2) double precision = E23.17

(This also applies to complex.)

Specification Statements

◆ The specification statements provide the compiler with information about the nature of the variable names in the source program. Specification statements may appear anywhere in the program but must precede the usage of the specified variables.

COMMON STATEMENTS

◆ The form of the COMMON statement is:

$$\text{COMMON} \quad \left[\begin{array}{c}\text{REAL}\\ \text{INTEGER}\\ \text{DOUBLE PRECISION}\\ \text{COMPLEX}\end{array}\right] \quad a(k_1),\ b(k_2),\ldots..$$

where a,b,... are variable or array names and $k_1,k_2,....$ are optional maximum subscript specifications not limited to any specific number of dimensions. The COMMON statement has the normal FORTRAN connotation; that is, variables that appear in COMMON in related programs and subprograms share the same positionally-oriented storage locations.

The following COMMON statements are valid for Interactive FORTRAN:

```
COMMON    A,B,C(10),I,J
COMMON    A(5,10),B(10),C(3,3),I
COMMON    REAL I(3,3)
COMMON    INTEGER A(5,5),B(100)
COMMON    INTEGER AA(3,2,2,2,2,2,2,2)
COMMON    DOUBLE PRECISION A(100),B(5,5)
COMMON    COMPLEX I (30), Z
```

TYPE STATEMENTS

◆ The forms of the Type statements are:

REAL	$a(k_1),b(k_2),\ldots.$
INTEGER	$a(k_1),b(k_2),\ldots.$
DOUBLE PRECISION	$a(k_1),b(k_2),\ldots.$
COMPLEX	$a(k_1),b(k_2),\ldots.$

where a,b,... are variable or array names and $k_1,k_2,....$ are optional maximum subscript specifications and are not limited to any specific number of dimensions.

If a subroutine name is used as an argument in a CALL statement it must be defined in the following way:

1. an F in column 1.
2. the names of the subroutines to be used as arguments from column 7 on.

TYPE STATEMENTS (Cont'd)

For Example:

```
Column 1        Column 7
   ↓               ↓
                   REAL X (10),Y(10)
   F               ZIP,JUST
                    .
                    .
                    .
                   CALL SUB2 (X(1),Y,ZIP,JUST)
                    .
                    .
                    .
                   END
                   SUBROUTINE ZIP (I)
   F               I
                   CALL I
                    .
                    .
                    .
                   END
                   SUBROUTINE JUST (J)
                    .
                    .
                    .
                   END
```

FORTRAN Supplied Mathematical Functions

◆ The following table shows the available library function routines with definitions:

Function	May be Written as	Definition
Exponential	EXP (Arg) EXPF (Arg)	e^{Arg}
Natural Logarithm	ALOG Arg LOGF ALOGF LOG	ln (Arg)
Sine	SIN SINF	sin (Arg)
Cosine	COS COSF	cos (Arg)
Hyperbolic Tangent	TANH TANHF	tanh (Arg)
Square Root	SQRT SQRTF	$(Arg)^{1/2}$
Absolute Value	ABS ABSF	Arg
Arctangent	ATAN (Arg) ATANF (Arg)	arctan (Arg)
Argument or Amplitude of a complex number	ARG (Arg) ARGF (Arg)	$\arctan\left(\frac{b}{a}\right)$
Absolute value or modulus of a complex number	MAG (Arg) MAGF (Arg)	$a + bi = (a^2 + b^2)1/2$

In all cases the argument specification determines the precision of the function.

Relative Characteristics of BTSS Interactive FORTRAN

Features	ASA	BTSS	TOS/TDOS
Max. Statement Number	99999	99999	99999
Max. Continuation Cards	19	None - each statement must be < 255 characters.	19
Specification statements must precede executable statements.	YES	NO	NO
Variable name - max. characters.	6	6	6
Assigned GO TO	YES	NO	YES
Logical IF	YES	YES	YES
Double Precision	YES	YES	YES
Complex Operations	YES	YES	YES
Max. - Array Dimensions	3	No Limit Common - cannot exceed product of 393 words. Type - cannot exceed product of 504 words.	7
Subscripts may be any unsigned integer constant, variable or expression.	NO	YES	NO

The following FORTRAN statements are not in BTSS Interactive FORTRAN:

ASSIGN	LOGICAL
BLOCK DATA	EQUIVALENCE
Labeled COMMON	DIMENSION
DATA	FUNCTION
Assigned GO TO	Statement Function

The following features are extensions available in BTSS Interactive FORTRAN:

- Basic functions may not be prefixed by D or C but may be suffixed with an F and will be accepted. The argument determines mode of computation.
- Debug features FLOW, TRACE, DUMP and PDUMP.
- Subroutines may contain recursive calls.
- Integer expressions with no limitations may be used whenever FORTRAN permits an integer variable.
- Abbreviations of D.P. for DOUBLE PRECISION and SUBR for SUBROUTINE are permitted.

DIFFERENTIAL EQUATIONS SUBROUTINE

SNODE (Spectra 70 Numerical Ordinary Differential Equations Subroutine)

◆ A general purpose subroutine for the numerical solution of a system of n simultaneous ordinary differential equations.

Calling Sequence

◆ CALL SNODE (X,Y,YP,P,DERIV,OUT,HALF,DUB)

The first parameter, X, is a double-precision variable containing the current value of the independent variable.

The next three parameters are double-precision storage arrays.

The last four parameters are names of user provided subroutines.

DERIV Subroutine

◆ The calling sequence of the subroutine is:

CALL DERIV (X,Y,YP).

It is written and provided by the user to calculate the values of the derivatives using the current values of the independent and dependent variables.

OUT Subroutines

◆ The calling sequence of the subroutine is:

CALL OUT (X,Y,YP,P).

It is written and provided by the user to be called each time a new point of the solution has been computed (see description of the parameter "all point switch").

HALF and DUB Subroutines

◆ The calling sequences are:

CALL HALF (X, Y, YP, P)
CALL DUB (X, Y, YP, P).

These are user provided subroutines which are called whenever a halving or doubling, respectively, of the step size is about to take place.

Storage Arrays

◆ The Y array is a block of n double precision words provided by the user for the storage of the current values of the dependent variables. These are set to the initial values before the calling SNODE.

The YP array is a block of n double precision words provided by the user for the storage of the current values of the derivatives of the dependent variables.

The P array is a block of (14 + 7n) double precision words provided by the user to accommodate the parameters and working storage.

Storage Arrays (Cont'd)

The parameters occupy the first 14 words of the P array. The first nine must be initialized by the user before the call.

They are:

P(1) Number of equations.

P(2) Step or interval size.

P(3) Number of figures of accuracy desired (may be any number).

P(4) Type of accuracy switch (zero--significant figure or relative accuracy, nonzero--decimal or absolute accuracy).

P(5) Endpoint switch (zero--continue to run, nonzero--hit endpoint).

P(6) Endpoint value (relevant only if P(5) is nonzero).

P(7) All point switch (zero--OUT called only for good points, nonzero--OUT called for all points).

P(8) Halving switch (zero--allow halving of the step size, nonzero--suppress halving).

P(9) Doubling switch (zero--allow doubling of the step size, nonzero--suppress doubling).

P(10) Maximum local error estimate (available to user after each step).

The remaining parameters P(11) - P(14), are not used in the simplified calling sequence.

The (7n) double precision words of working storage (beginning with the 15th word of P contain, for each step, the following:

Location in Working Storage	Contents at m-th Step
7i - 6	$p_m - c_m$ (predicted value minus corrected value)
7i - 5	Y_{im}
7i - 4	Y^1_{im} (back derivatives)
7i - 3	$Y^1_i(m-1)$
7i - 2	$Y^1_i(m-2)$
7i - 1	$Y^1_i(m-3)$
7i	Y_{io} (initial conditions)

STOP Subroutine

◆ A STOP subroutine is provided with SNODE which prints out, upon the occurrence of an error, a message describing the type of error.

Sample Problem

```
C     F.W. SCHNEIDER
C     EXAMPLE 1
C     TWO SIMULTANEOUS EQUATIONS USING SNODE
      DOUBLE PRECISION X,Y(2),YP(2),P(28)
C     P ARRAY CONTAINS (14+2*7) DOUBLE WORDS
F     DER,OUT,HALF,DUB
C     SET UP FOR TWO EQUATIONS WITH 5 FIGURES OF DECIMAL ACCURACY
      P(1)=2
      P(3)=5
      P(4)=1
C     START AT 0 WITH STEP SIZE 0.05
      X=0
      P(2)=.05
C     HIT ENDPOINT PI
      P(5)=1
      P(6)=3.1415926535897932
C     PRINT GOOD POINTS ONLY, ALLOW HALVING AND DOUBLING
      P(7)=0
      P(8)=0
      P(9)=0
      P(10)=0
C     INITIALIZE DEPENDENT VARIABLES (Y(1) IS COS, Y(2) IS SIN)
      Y(1)=1
      Y(2)=0
   50 FORMAT ('1 EXAMPLE 1'/'0X,Y1,Y2'/' YP1,YP2,MAXER')
      PRINT 50
      CALL SNODE (X,Y,YP,P,DER,OUT,HALF,DUB)
      STOP
C
      SUBROUTINE DER (X,Y,YP)
      DOUBLE PRECISION X,Y(2),YP(2)
      YP(1)=-Y(2)
      YP(2)=Y(1)
      RETURN
      END
C
      SUBROUTINE OUT (X,Y,YP,P)
      DOUBLE PRECISION X,Y(2),YP(2),P(28)
   50 FORMAT (1X,3E24.17)
      PRINT 50,X,Y(1),Y(2),YP(1),YP(2),P(10)
      RETURN
      END
C
      SUBROUTINE HALF (X,Y,YP,P)
      DOUBLE PRECISION X,Y(2),YP(2),P(28)
   51 FORMAT(' INTERVAL HALVED')
      PRINT 51
      RETURN
      END
C
      SUBROUTINE DUB (X,Y,YP,P)
      DOUBLE PRECISION X,Y(2),YP(2),P(28)
   51 FORMAT(' INTERVAL DOUBLED')
      PRINT 51
      RETURN
      END
```

Sample Problem (Cont'd)

```
C
      SUBROUTINE SNODE (X,Y,YP,P,D,O,H,B)
F     D,O,H,B,STOP
      DOUBLE PRECISION X,Y(2),YP(2),P(28)
      P(11)=0.0
      P(12)=1.0
      P(13)=0.0
      P(14)=0.0
      CALLODESTA(X,Y,YP,P,D,D,O,H,B,STOP,ODENPT,ODENDR,ODEERR,ODERUN)
      P(12)=0.0
      CALLODERUN(X,Y,YP,P,D,D,O,H,B,STOP,ODENPT,ODENDR,ODEERR,ODESTA)
      RETURN
      END
C
      SUBROUTINE STOP (X,Y,YP,P)
      DOUBLE PRECISION X,Y(2),YP(2),P(28)
   51 FORMAT ('ODEPENDENT VARIABLE ZERO, RELATIVE ACCURACY')
   52 FORMAT ('OINTERVAL TOO SMALL--X+H=X')
   53 FORMAT ('OTHIRTY CONSECUTIVE INTERVAL REDUCTIONS')
   54 FORMAT ('ONUMBER OF EQUATIONS ZERO OR NEGATIVE')
   55 FORMAT (' JOB TERMINATED')
      N=P(11)
      IF(N.EQ.1) PRINT 51
      IF(N.EQ.2) PRINT 52
      IF(N.EQ.3) PRINT 53
      IF(N.EQ.4) PRINT 54
      PRINT 55
      STOP
      END
```

Sample Problems

```
C   N O T E :
C    THE FOLLOWING EXAMPLES DO NOT DIFFERENTIATE
C    BETWEEN SYSTEM AND USER RESPONSE SINCE THEY
C    ARE A REPRODUCTION OF AN ACTUAL TERMINAL SESSION
```

```
        /CATALOG
   GRACE      TIC        DATE       TAPE02
  *TAPE25     TAPE08     TAPE01     TAPE02
*- SIGNIFIES LOCKED FILE
READY
/DROP GRA
DELETED
/PRINT# GRACE
     10         D.P. C0,C1,T0,T1,SLOPE,PRES,CAP,CK,BP,WVT
     20 10      FORMAT(9E15.5)
     25 15      CONTINUE
     30         READ 10,C0,C1,T0,T1,PRES,CAP,CK,BP
     40         SLOPE=(C0-C1)/(T0-T1)
     50         WVT=(SLOPE*PRES*CAP*CK)/BP
     60 20      FORMAT(8H SLOPE =E15.5,5X   5HWVT =E15.5)
     70         PRINT 20,SLOPE,WVT
     75 30      FORMAT(///////)
     76         PRINT30
     80         GOTO15
     90         STARTI0
```

```
/CODE
C0 = 1.44
C1 = 23
T0 = 44,
T1 = 55.7
PRES = 122.5
CAP = 12.7
CK = 44.2
BP = 2
SLOPE =  .18427350427350418E 01      WVT =  .6335705444444432E 05
```

```
C0 = 1.44,23,44,55.7,122.5,12.7,44.2,2
SLOPE =  .18427350427350418E 01      WVT =  .6335705444444432E 05
```

```
    5 C      THIS PROGRAM CALCULATES THE DAY OF THE WEEK
    6 C           FOR ALL DATES FROM 1801-2099
   10 10     FORMAT(1X2I5)
   20        READ10,ITAPE
   30 12     FORMAT(1XA8)
   40        D.P. DOW(7)
   45        REWIND ITAPE
   50        DO 13 I=1,7
   60        READ (ITAPE,12) DOW(I)
   70 13     CONTINUE
   80        INTEGER DAY,YEAR,MO(12),MAXDAY(12)
   90        DO 11 I=1,12
  100        READ(ITAPE,10) MO(I),MAXDAY(I)
  110 11     CONTINUE
  120 30     FORMAT(5X,12HENTER A DATE)
  130        PRINT 30
  140 45     READ 10 ,MONTH
  150        IF(MONTH-12) 50,50,60
  160 50     IF(MONTH-1) 60,70,70
  170 20     FORMAT(5X,13HINVALID MONTH)
  180 60     PRINT 20
  190        GO TO 45
  200 70     READ 10, DAY
  210        IF(DAY-1) 80,90,90
  220 75     FORMAT(5X11HINVALID DAY)
  230 80     PRINT 75
  240        GO TO 70
  250 90     IF(DAY-MAXDAY(MONTH)) 100,100,80
  260 100    READ 10 YEAR
  270        IF(YEAR-100)155,110,110
  280 110    IF(YEAR-1800)130,130,140
  290 115    FORMAT(5X27HYEAR NOT IN RANGE 1801-2099)
  300 130    PRINT 115
  310        GO TO 100
  320 140    IF(YEAR-2099)150,150,130
  330 150    IF(YEAR-1900)151,152,152
  340 151    IYR=18
  350        YEAR=YEAR-1800
  360        GO TO 160
  370 152    IF(YEAR-2000)153,154,154
  380 153    IYR=19
  390        YEAR=YEAR-1900
  400        GO TO 160
  410 154    IYR=20
  420        YEAR=YEAR-2000
  423        GOTO160
  424 155     IYR=19
  425 160     JYEAR=YEAR
  430 260     JYEAR=JYEAR-12
  440        IF(JYEAR)161,162,260
  450 161     JYEAR=JYEAR+12
  460 162    IY=YEAR/12+JYEAR+JYEAR/4
  470 163    IY=IY-7
  480        IF(IY)164,165,163
  490 164    IY=IY+7
  500 165    IY=IY+MO(MONTH)+DAY
  510 166    IY=IY-7
  520        IF(IY)167,168,166
  530 167    IY=IY+7
  535        GOTO 169
  540 168    IY=7
```

```
545 169      IF(YEAR)178,17,178
546 178          IF(MONTH-2) 177,177,170
547 177          KY=JYEAR+4
550 269      KY=KY-4
560          IF(KY)310,171,269
570 171      IF(IY-1) 172,173,174
580 172      CONTINUE
590          STOP
600 173      IY=7
610          GO TO 170
620 174      IY=IY-1
630 175      FORMAT(///16H DAY OF WEEK FOR,I4,1H/,I2,1H/,I4,3H =
        ,A8,///)
640 170      GO TO 370
650 399       IYR=IYR+YEAR
660          PRINT 175,MONTH ,DAY,IYR,DOW(IY)
670            GOTO 11
680          STOP
690 17       IF(IYR-19) 172,171,178
700 370      IF(IYR-19)91,99,92
710 91       IY=IY+2
720          GO TO 93
730 92       IY=IY-1
740 93       IF(IY)94,95,96
750 94       IY=IY+7
760          GO TO99
770 95       IY=7
780          GO TO 99
790 96       IF(IY-7) 99,99,97
800 97       IY=IY-7
810 99       IYR=IYR*100
820          GO TO 399
830 310      IF(MONTH-2)170,311,172
840 311      IF(DAY-28)170,170,313
850 312      FORMAT(/27H 2/29 INVALID-NOT LEAP YEAR)
860 313      PRINT 312
870          GO TO 45
```

```
/PRINT# TAPE25
   0 CT SENTINEL
  10 S U N .
  20 M O N .
  30 T U E S
  40 W E D S
  50 T H U R
  60 F R I .
  70 S A T .
  80 1,31
  90 4,29
 100 4,31
 110 0,30
 120 2,31
 130 5,30
 140 0,31
 150 3,31
 160 6,30
 170 1,31
 180 4,30
 190 6,31
 200
```

```
      /
DELETED
/CODE DATE
   880        *START*10
ITAPE = 25
     ENTER A DATE
MONTH = 12
DAY = 25
YEAR = 66


DAY OF WEEK FOR  12/25/1966 = S U N .




     ENTER A DATE
MONTH = 2
DAY = 29
YEAR = 67

2/29 INVALID-NOT LEAP YEAR
MONTH = 2
DAY = 29
YEAR = 68


DAY OF WEEK FOR   2/29/1968 = T H U R




     ENTER A DATE
MONTH = 2
DAY = 29
YEAR = 1817

2/29 INVALID-NOT LEAP YEAR
MONTH = 2
DAY = 29
YEAR = 1816


DAY OF WEEK FOR   2/29/1816 = T H U R
```

```
    ENTER A DATE
MONTH =
RE-ENTER
9
DAY = 23
YEAR = 77

DAY OF WEEK FOR   9/23/1977 = F R I .


    ENTER A DATE
MONTH =
```

6. SWITCHING MODULE

DESCRIPTION

◆ The Switching Module is a component of the Basic Time Sharing System that enables a customer of BTSS to avail himself of the batched processing facilities in the Tape Operating System. The Switching Module performs two distinct functions:

1. it directs the transition from the time sharing mode to a batch processing mode
2. it directs the transfer of data from tape to disc when the time sharing system is initiated after the completion of batch processing under TOS.

Prior to the initiation of the Switching Module and the subsequent transfer to TOS the time sharing customer stores the data in his own disc files to be processed under Monitor control. In addition, he is required to specify certain Switching Module commands that indicate the disposition of the data before and after the Monitor session. These commands must be inserted into a file that the customer creates and that has the name JBSTREAM. When the time sharing session terminates the Switching Module, operating in accordance with these stored commands, prepares a Monitor job stream tape (SYSIPT) for the customer, transfers any data associated with the job stream to an available work tape, and makes provision for the future conversion of data from tape to the customer's disc file at the next initiation of a time sharing operation. Upon completion of all Switching Module functions, the processor is released for a Monitor session.

INITIATION OF TIME SHARING

◆ After the BTSS is loaded and initiated, control is given to the Switching Module which then determines if a Monitor operation has preceded it and if monitor output is to be used during the impending time sharing session. When required, data is transferred from tape to the time sharing customer's disc files. The Switching Module then passes control to the BTSS Executive and Time Sharing commences.

SWITCHING MODULE COMMAND FILE

◆ All commands to the Switching Module must be stored by the time sharing customer in a special file named JBSTREAM. This file may also be used for the storage of data that is to be transferred to tape for Monitor processing. If the file contains both commands and data the commands must appear before the data in the file. Each Switching Module command must be inserted in the file as a separate record. The file name (JBSTREAM) is the same for all customers and there may be as many JBSTREAM files as customers.

SWITCHING MODULE COMMANDS

Input Command

◆ The format of the Input Command is as follows:

**INPUTΔfilename [,filename] [,filename] [,filename]

The Input Command informs the Switching Module that the data contained in the specified disc file(s) is to be transferred to the Monitor input tape (SYSIPT) for processing in a subsequent TOS operation. A maximum of 26 files can be transferred. Data is transferred to tape as 80-byte card images batched four to a block. Records that are not 80 bytes long, are expanded to the proper size. Those exceeding 80 bytes are truncated. Transfer of data terminates when an end-of-file condition is detected on the disc. All Monitor control statements such as //ΔASSGN and //ΔEXEC must be included in the customer's data file with the exception of //ΔSTARTM and //ΔENDMON. The latter are supplied by the Switching Module.

In the event that multiple customer files are specified in this command, data is transcribed to tape in the same sequence in which the files are named.

Example:

**INPUTΔJBSTREAM,ASYSINPT

Note:

In the case of JBSTREAM, the transfer of data begins with the first record following the last Switching Module command.

Data Command

◆ The format of the Data command is as follows:

**DATAΔfilelabel,filename [,filename] [,filename] [,filename]

The Data command informs the Switching Module that the data contained in the specified disc file(s) is not part of the Monitor job stream and that it is to be transferred to an available work tape rather than the Monitor input tape (SYSIPT). A maximum of 26 files can be transferred. The tape is created as a standard-labeled, single-file, single-reel volume. The eight byte filelabel field of the command is used as the first eight bytes of the label identification field in all labels written to the tape. It is recommended that the first five bytes of the usercode be used as the first five bytes of the filelabel to facilitate the identification of tapes.

Data is transferred from the disc to tape without modification. Each record in the disc file becomes a block of data on the tape.

If multiple customer files are specified in this command, the files are transcribed to tape in the same sequence in which they are named.

Example:

**DATAΔRCA00FL1,TAPEFILE

Note:

The data contained in TAPEFILE is transferred to tape without modification. The tape will have standard Spectra 70 labels and the first eight characters of the label identification field are RCA00FL1.

Output Command

- The output command has the following format:

**OUTPUTΔfilelabel,filename [,filename] [,filename] [,filename]

The Output command informs the Switching Module that the named disc files are available to receive data that is to be transcribed from tape when the next time sharing session is initiated. The eight character filelabel field specifies the first eight bytes of the label identification field of the tape file. A maximum of 26 files can be transferred.

All tape files to be transferred to disc must be single-file, single-reel volumes with standard Spectra 70 labels.

The specified disc files to which the data is to be transferred must be created in a time sharing session prior to the one in which the transfer will be executed.

Data is transcribed to the disc in single record format with sequence numbers that are increments of ten. Each record on the disc corresponds to a block of data on tape.

The maximum tape block size that can be transferred to the disc without modification is 255 bytes. The remainder is truncated.

Example:

**OUTPUTΔRCA00FL2,DISCFILE

Note:

The tape file that has RCA00FL2 in the first eight position of its label identification field, will be transferred to DISCFILE when the next time sharing session is initiated.

End Command

- The End command has the following format:

**END

This command informs the Switching Module that no more commands follow it in the JBSTREAM file.

APPENDIX A

LANGUAGE SUMMARY

◆ The legal BTSS commands are:

/CODE [Δfilename]

/EDIT [Δfilename]

- SYMBOL character
- CHECK
- NOCHECK
- RESUME [{C / or / N}]
- LOOP
- ENDLOOP
- HELP

} Processor Commands

- sequence-listΔCHANGEΔstring-1ΔTOΔstring-2
- sequence-listΔCHANGEALLΔstring-1ΔTOΔstring-2
- sequence-listΔDELETE [Δstring]
- sequence-listΔDELETEALLΔstring
- sequence-listΔFINDΔstringΔ{DROPP / DROPS}
- sequence-listΔFINDΔstring-1Δ{REPLACEP / REPLACES}Δstring-2
- sequence-listΔFINDΔstring-1Δ{INSERTP / INSERTS}Δstring-2
- sequence-listΔFINDALLΔstring-1Δ{INSERTP / INSERTS}Δstring-2
- sequence-listΔPREFIXΔstring
- sequence-listΔSUFFIXΔstring
- sequence-list [ΔPRINT]
- sequence-list [ΔPRINTΔstring]
- sequence-listΔstring
- sequence-listΔTEXTΔstring
- sequence-list-1ΔCOPYΔsequence-list-2
- sequence-list-1ΔWASΔsequence-list-2
- sequence-listΔSETPCΔstring

} Record Commands

LANGUAGE SUMMARY (Cont'd)

/IN [#] [Δfilename]

/CATALOG

/DROPΔfilename

/DOΔfilename [Δsequence # [ΔABCD]]

/PRINT [#] [Δfilename] [Δsequence# [ΔTOΔsequence#]]

/RESEQ [Δfilename] [Δsequence#]

/LOCK [Δfilename] ΔASΔfilename

/UNLOCKΔfilenameΔASΔfilename

/OPTION [{N / P}] [Δfilename]

/SAY [Δmessage]

/ONΔusercode [1-3 symbols] [Δ{P / N / OPTIONP / OPTIONN / OPTION}]

/OFF

/COST

/HELP

/ENDO

/RENAME [Δfilename] ΔASΔfilename

LANGUAGE SUMMARY (Cont'd)

**INPUTΔfilename
**DATAΔfilelabel,filename
**OUTPUTΔfilelabel,filename
**END
} Switching Module

/CODE [Δfilename]

EXECUTEΔ [ON / OFF]

SAVEΔ [ON / OFF]

CALCULATOR

*

START [blank / Δline number / Δ*sequence number]

HALT [blank / Δline number / Δ*sequence number]

RESET [blank / Δline number / Δ*sequence number]

PDUMPΔlist

DUMPΔlist

TRACEΔ [ON / OFF] list

FLOWΔ [ON / OFF]

APPENDIX B

EDIT ERROR MESSAGES

INPUT ERRORS

BAD INPUT (#01)
RESUME NOT VALID HERE

BAD INPUT (#02)
NON-BLANKS FOUND AFTER COMMAND

BAD INPUT (#03)
ONLY ONE CHAR ALLOWED IN STRING FOLLOWING SETPC

BAD INPUT (#04)
STATEMENT TOO LARGE FOR LOOP STORAGE

BAD INPUT (#05)
NON-BLANK SYMBOL EXPECTED

BAD INPUT (#06)
IN LOOP MODE, BUT FLOATING SEQ NOT ALLOWED HERE

BAD INPUT (#07)
OFFSET TOO LARGE FLOATING NUMBER CAN BE 999999

BAD INPUT (#08)
NUMERIC CHAR EXPECTED

BAD INPUT (#09)
BLANK, COMMA, OR DASH EXPECTED

BAD INPUT (#10)
RANGE (AND SINGLE) SEQ NUM(S) MUST FOLLOW LOOP

BAD INPUT (#11)
MORE THAN 8 ELEMENTS IN LEFT SEQ LIST

BAD INPUT (#12)
MORE THAN 15 ELEMENTS IN RIGHT SEQ LIST

BAD INPUT (#13)
FLAT OR INVERTED RANGE GIVEN

BAD INPUT (#14)
SEQ NUMBER EXCEEDS 6 DIGITS

BAD INPUT (#15)
SEQ LIST EXPECTED

INPUT ERRORS (Cont'd)

BAD INPUT (#16)
INVALID COMMAND CODE (OR DELIMITER EXPECTED)

BAD INPUT (#17)
NON-BLANK FOLLOWS STRING 2 OR NULL STRING 1

BAD INPUT (#18)
THE VERB GIVEN EXPECTS A DIFFERENT # OF STRINGS

BAD INPUT (#19)
CHANGE VERB EXPECTS TO BETWEEN THE 2 STRINGS

BAD INPUT (#20)
INSERTP OR INSERTS EXPECTED

BAD INPUT (#21)
INSERTP, INSERTS, REPLACEP, OR REPLACES EXPECTED

BAD INPUT (#22)
DROPP OR DROPS EXPECTED

BAD INPUT (#23)
A SUFFIX P OR S IS EXPECTED

BAD INPUT (#24)
EXTRA NON-BLANKS FOUND

BAD INPUT (#25)
FLOATING SEQ-NUM EXPECTED

LIST OF EXECUTION ERROR MESSAGES

◆ Depending on the error case one of the following nine sets of messages is typed out:

1)
COPY PROCESS NOT PERFORMED-RECORD WOULD EXCEED 255 CHARS.

2)
WAS PROCESS NOT PERFORMED-RECORD WOULD EXCEED 255 CHARS.

3)
COPY PROCESS NOT EXECUTED-MISSING RECORDS ABOVE.
TO CONTINUE COPY PROCESS WITHOUT MISSING RECORD-TYPE RESUME.

4)
WAS PROCESS NOT EXECUTED-MISSING RECORDS ABOVE.
TO CONTINUE WAS PROCESS WITHOUT MISSING RECORD-TYPE RESUME.

5)
REVISED LENGTH FOR RECORD XXXXXX EXCEEDS 255 CHARS-RECORD UNCHANGED.

6)
REVISED LENGTH FOR RECORD XXXXXX EXCEEDS 255 CHARS-RECORD UNCHANGED.
TYPE RESUME TO SKIP RECORD AND CONTINUE.

7)
EXECUTING LOOP STATEMENT XX. INDEX IS XXXXXX.
COPY PROCESS NOT PERFORMED-RECORD WOULD EXCEED 255 CHARS.

8)
EXECUTING LOOP STATEMENT XX, INDEX IS XXXXXX.
REVISED LENGTH FOR RECORD XXXXXX EXCEEDS 255 CHARS-RECORD UNCHANGED.

9)
EXECUTING LOOP STATEMENT XX. INDEX IS XXXXXX.
REVISED LENGTH FOR RECORD XXXXXX EXCEEDS 255 CHARS-RECORD UNCHANGED.
TYPE RESUME TO SKIP RECORD AND CONTINUE.

Inputting a null message after any of the above messages sets will return the appropriate message 10.) or 11.). These will be followed by the record itself.

10)
PARTIALLY GENERATED RECORD IS -

11)
UNMODIFIED RECORD XXXXXX HAS LENGTH XXX.

LIST OF ABNORMAL MESSAGES

◆ The following messages are issued when appropriate, regardless of the current CHECK/NOCHECK status. However, these messages are never issued during LOOP operation.

1)
SEQ NUMBER XXXXXX UNDEFINED

2)
SEQ NUMBER XXXXXX UNCHANGED

3)
SEQ RANGE XXXXXX TO XXXXXX UNDEFINED

4)
SEQ RANGE XXXXXX TO XXXXXX UNCHANGED

LIST OF MESSAGES SUPPLIED AFTER USER STOPS

◆ Stops during LOOP execution first supplies either 1.) or 2.). If message 2. is supplied, the process has further incremented the LOOP index in preparation for another pass through the statement.

1)
STOPPED AT LOOP STATEMENT XX. INDEX IS XXXXXX.

2)
STOPPED AT LOOP, INDEX IS XXXXXX.

3)
STOPPED DURING GENERATION OF COPY RECORD USING SEQ # XXXXXX.

4)
STOPPED DURING DELETE PHASE OF WAS PROCESS
RECORD XXXXXX FROM RANGEXXXXXX TO XXXXXX LAST DELETED

5)
STOPPED DURING DELETE PHASE OF WAS PROCESS
LAST SEQ LIST ELEMENT PROCESSED WAS XXXXXX TO XXXXXX

6)
STOPPED DURING DELETE PHASE OF WAS PROCESS
LAST SEQ LIST ELEMENT PROCESSED WAS XXXXXX

7)
SEQ # XXXXXX LAST PUT INTO FILE

8)
ABOVE RECORD WAS LAST ONE PROCESSED

9)
ABOVE SEQ LIST ELEMENT LAST ONE PROCESSED

10)
UNDEFINED SEQ RANGE XXXXXX TO XXXXXX LAST PROCESSED

11)
UNDEFINED RECORD XXXXXX LAST PROCESSED

12)
SEQ NUMBER XXXXXX LAST ONE DELETED

13)
LAST SEQ LIST ELEMENT PROCESSED WAS UNDEFINED XXXXXX TO XXXXXX

14)
LAST SEQ LIST ELEMENT PROCESSED WAS UNDEFINED XXXXXX

LIST OF MESSAGES SUPPLIED AFTER USER STOPS (Cont'd)

15)
RECORD # XXXXXX WAS LAST ONE PROCESSED

16)
PARTIALLY FORMED RECORD IS-

17)
GENERATED RECORD IS-

18)
GENERATED RECORD IS NULL

19)
NO MORE ANALYSIS AVAILABLE-RESUME CONTINUES

20)
TYPE RESUME TO CONTINUE THIS PROCESSING