DISCLAIMER

The MTS Manual is intended to represent the current state of the Michigan Terminal System (MTS), but because the system is constantly being developed, extended, and refined, sections of this volume will become obsolete. The user should refer to the computing center Newsletter, Computing Center Memos, and future Updates to this volume for the latest information about changes to MTS.

Copyright 1983 by the Regents of the University of Michigan. Copying is permitted for nonprofit, educational use provided that (1) each reproduction is done without alteration and (2) the volume reference and date of publication are included. Permission to republish any portions of this manual should be obtained in writing from the Director of the University of Michigan Computing Center.

## PREFACE

The software developed by the Computing Center staff for the operation of the high-speed processor computer can be described as a multiprogramming supervisor that handles a number of resident, reentrant programs. Among them is a large subsystem, called MTS (Michigan Terminal System), for command interpretation, execution control, file management, and accounting maintenance. Most users interact with the computer's resources through MTS.

The MTS Manual is a series of volumes that describe in detail the facilities provided by the Michigan Terminal System. Administrative policies of the Computing Center and the physical facilities provided are described in a separate publication entitled Introduction to the Computing Center.

The MTS volumes now in print are listed below. The date indicates the most recent edition of each volume; however, since volumes are updated by means of CCMemos, users should check the Memo List, copy the files *CCMEMOS or *CCPUBLICATIONS, or watch for announcements in the Computing Center Newsletter, to ensure that their MTS volumes are fully up to date.

```
    Volume 1: The Michigan Terminal System, January 1984
    Volume 2: Public File Descriptions, April }198
    Volume 3: System Subroutine Descriptions, April 1981
    Volume 4: Terminals and Networks in MTS, March 1984
    Volume 5: System Services, May 1983
    Volume 6: FORTRAN in MTS, October 1983
    Volume 7: PL/I in MTS, September 1982
    Volume 8: LISP and SLIP in MTS, June 1976
    Volume 9: SNOBOL4 in MTS, September 1975
    Volume 10: BASIC in MTS, December 1980
    Volume 11: Plot Description System, August }197
    Volume 12: PIL/2 in MTS, December 1974
    Volume 13: The Symbolic Debugging System, September }198
    Volume 14: 360/370 Assemblers in MTS, May 1983
    Volume 15: FORMAT and TEXT360, April }197
    Volume 16: ALGOL W in MTS, September 1980
    Volume 17: Integrated Graphics System, December 1980
    Volume 18: The MTS File Editor, August }198
    Volume 19: Tapes and Floppy Disks, February 1983
```

Other volumes are in preparation. The numerical order of the volumes does not necessarily reflect the chronological order of their appearance; however, in general, the higher the number, the more specialized the volume. Volume 1, for example, introduces the user to

MTS 5: System Services

Page Revised September 1985

MTS and describes in general the MTS operating system, while Volume 10 deals exclusively with BASIC.

The attempt to make each volume complete in itself and reasonably independent of others in the series naturally results in a certain amount of repetition. Public file descriptions, for example, may appear in more than one volume. However, this arrangement permits the user to buy only those volumes that serve his or her immediate needs.

Richard A. Salisbury
General Editor

## PREFACE TO REVISED VOLUME 5

The May 1983 revision reflects the changes that have been made to MTS since the April 1980 edition.

The sections "IOH," "Extensions to the Amdahl 470 Operations," and "Extensions to the System/360 Model 67 Operations" have been moved into MTS Volume 14, $360 / 370$ Assemblers in MTS.

Acknowledgments for the descriptions and programs contained in this volume are as follows:

The SORT program was developed by Douglas Burr Smith to test several concepts relating to sorting algorithms and page faults in a large virtual memory environment. Additional development of both the program and the description in this volume has been done by Charles F. Engle.

The dynamic loader program and its description were originally produced by Ronald J. Srodawa. Recent additions to the program and description were made by Kenneth DeJong and Pat Sherry.

The linkage editor program and its description were produced by Kenneth DeJong, James Henriksen, and Pat Sherry.

The object-file editor program and its description were produced by Pat Sherry.

The ACCOUNTING command mode was originally developed as the program *PROJECTACCOUNT by Charles F. Engle. His continuing development of the program has led to the present version of ACCOUNTING which is described in this volume.

The remainder of the descriptions in this volume were either produced by or heavily modified from other documentation by the editorial staff.

MTS 5: System Services

May 1983
Preface . . . . . . . . . . . . 3
Preface to Revised Volume 5 . . 5
The SORT Utility Program ..... 11
Simple Character Sorts ..... 13
Elementary Sorting of
FORTRAN Output ..... 15
Notation ..... 16
The Control Statement ..... 17
The Mode's Description ..... 18
The Collating Sequence's Description ..... 18
The Defined Sequences'
Descriptions ..... 20
The Data Sets'
Descriptions ..... 21
The END Parameter ..... 25
Additional Parameters ..... 25
Restricted Parameters ..... 29
Optional Exits ..... 29
Exit SORTEO ..... 30
Exit SORTE1 ..... 31
Exit SORTE2 ..... 33
Exit SORTE3 ..... 35
Exit SORTE4 ..... 36
Exit SORTE5 ..... 37
Exit SORTE6 ..... 39
Exit SORTE7 ..... 40
Exit SORTE8 ..... 40
Exit SORTE9 ..... 41
Optional Entrance ..... 41
Entrance SORTEA ..... 41
Sort Timing Estimates ..... 42
Sort Optimization ..... 48
Number and Types ofCollating Fields48
Distribution of the Source Data ..... 48
Blocking Factors of Data
Sets ..... 49
Number, Type, and
Efficiency of Exit ..... 49
System Load ..... 49
Processing Magnetic Tapes ..... 50
Statistics and Diagnostics ..... 51
Sort Examples ..... 52
The SORT Subroutine ..... 57
Introduction ..... 57
Return Codes ..... 57
The Entry Point SORT ..... 57
The Entry Point SORT2 ..... 62
The Entry Point SORT3 ..... 65
The Entry Point SORT4 ..... 67
Bibliography ..... 71
Appendix A: Collating Fields ..... 72
Appendix B: Record77
Appendix C: Summary of theControl Statement81
The Dynamic Loader ..... 83
Elementary Loader Topics ..... 83
Introduction ..... 83
Overview of the Loading
Process ..... 83
Structure of
Compiler-Generated Object Modules ..... 85
Processing an Object
Module ..... 87
Storage Allocation During the Loading Process ..... 88
Advanced Loader Topics ..... 88
Modifying Programs inObject-Module Form88
Common-Section Processing ..... 93
Pseudo-Register Processing ..... 93
Merging External Symbols ..... 94
Library Processing ..... 95
Dynamic Loading ..... 98
Initial ESD Lists and
Low-Core Symbol Tables ..... 100
Appendix A: Entry Point
Determination ..... 104
Appendix B: Loader Error Messages ..... 105
Appendix C: Sample Loader
Map ..... 110
Appendix D: Loader RecordFormat113
Translator-Generated LoadRecords113
User-Generated Load
Records ..... 137
Library Control Records ..... 143
Virtual Memory Management ..... 147
The Linkage Editor ..... 159
Linkage Editor Command Language ..... 162
General Syntax ..... 162
Linkage Editor Commands ..... 164
ADD ..... 165
ALIAS ..... 165
ATTRIBUTE ..... 165
BLAST ..... 166
CLEAR ..... 166
COMBINE ..... 167
COMMENT ..... 167
COPY ..... 168
CSECT ..... 168
DELETE ..... 168
DISPLAY ..... 169
DUMP ..... 170
INCLUDE ..... 170
LIST ..... 170
MAP ..... 172
MCMD ..... 172
MODIFY ..... 172
MTS ..... 174
PUNCH ..... 174
PURGE ..... 174
RENAME ..... 175
REPLACE ..... 175
RETURN ..... 176
SCAN ..... 176
SET ..... 177
STOP ..... 178
UNLINK ..... 178
UPDATE ..... 179
XREF ..... 179
Command Modifiers ..... 180
A ..... 180
BC ..... 181
CHECK ..... 181
COMSAVE ..... 181
CSECT ..... 182
CSECT= ..... 182
EMPTY ..... 182
ENTRY= ..... 182
ENTRY ..... 183
FULL ..... 183
GAPSIZE ..... 183
GENSAVE ..... 184
LENGTH ..... 184
MISCSAVE ..... 184
MSGSAVE ..... 185
NAME ..... 185
NV ..... 185
OM ..... 185
ORL ..... 186
SLI ..... 186
SORT ..... 186
SYMSAVE ..... 187
TERSE ..... 187
TYPE ..... 187
V ..... 188
VERBOSE ..... 188
WXTOER ..... 188
Linkage Editor Example ..... 189
Principles of Operation ..... 192
Optimization of Object Modules ..... 193
Optimizations
Automatically Performed ..... 193
Optimizations Performed by the COMBINE Command ..... 195
Optimizations Performed by the PURGE Command ..... 196
Optimizations Obtained bythe SLI Modifier197
Optimizations Obtained by the BC Modifier ..... 197
Processing Special Records ..... 197
Attention-Interrupt
Processing ..... 199
Input Conversion ..... 200
Output Conversion ..... 204
Object-Module Naming Conventions ..... 205
The Object-File Editor ..... 207
Object-File Editor Command
Language ..... 212
Object-File Editor
Commands ..... 215
ADD ..... 215
CLEAR ..... 215
CLOSE ..... 215
COMIENT ..... 216
CREATE ..... 216
CSECT ..... 217
DELETE ..... 217
DISPLAY ..... 217
EDIT ..... 218
EMPTY ..... 219
EXPLAIN ..... 219
HELP ..... 220
INCLUDE ..... 220
LIST ..... 220
MAP ..... 222
MCMD ..... 222
MODIFY ..... 222
MTS ..... 223
PATCH ..... 224
PUNCH ..... 225
RENAME ..... 225
REPLACE ..... 225
RETURN ..... 226
SCAN ..... 226
SET ..... 227
SNIFF ..... 228
STOP ..... 228
UPDATE ..... 229
XREF ..... 229
Command Modifiers ..... 230
BREAK ..... 230
COMGEN ..... 230
COMSAVE ..... 230
DEF ..... 230
DEFSAVE ..... 231
DIRECTORY ..... 231
DLR ..... 231
DMD ..... 231
EMPTY ..... 231
FULL ..... 232
GAPSIZE ..... 232
LENGTH ..... 232
LIBRARY ..... 232
MISCSAVE ..... 233
MSGSAVE ..... 233
OPTIMIZE ..... 233
ORL ..... 233
REPGEN ..... 233
REPSAVE ..... 234
SLOTS ..... 234
SORT ..... 234
SYMSAVE ..... 234
TERSE ..... 235
TYPE ..... 235
UNDEF ..... 236
VERBOSE ..... 236
VERIFY ..... 236
Object-File Editor Example ..... 237
ACCOUNTING ..... 239
Introduction ..... 239
Displaying StatusInformation about anIndividual Signon ID . . . . 239
The ACCOUNTING MANAGEMENT ..... 241
The Commands ..... 242
Adding Money to a Signon ..... ID ..... 243
Subtracting Money from a
Signon ID ..... 243
Modifying Money for a Signon ID ..... 244
Equalizing Money for a Signon ID ..... 244
Expiring a Signon ID ..... 244
Deleting a Signon ID ..... 245
Displaying a Signon ID ..... 246
Continuing with a
Different Signon ID ..... 246
Other Signon Ranges ..... 248
Blocks ..... 248
ENTIRE ..... 248
Signon-Range Groups ..... 249
Defined Signon-ID Ranges ..... 249
PROJECT ..... 250
The Keywords ..... 252
Changing Maximum Charge ..... 252
Changing Maximum Disk Space ..... 252
Changing Expiration Time ..... 252
Changing Maximum Concurrent Signons ..... 254
Changing Maximum Terminal Time ..... 255
Changing Maximum Plotting Time ..... 256
Filtering Signon IDs
within a Signon Range ..... 257
Changing NOCHANGE ..... 257
Changing Normal Priority
Access ..... 259
Changing Low Priority
Access ..... 260
Specifying a Project
Signon File ..... 260
Inhibiting Project SignonFile Attentions261
Resetting Passwords ..... 262
Controlling the Listing ..... 263
Producing Headings ..... 264
Other Commands ..... 265
The SET Command ..... 265
Temporarily Returning to
MTS ..... 265
Terminating ACCOUNTING
MANAGEMENT ..... 266

MTS 5: System Services

Page Revised September 1985
Miscellaneous . . . . . . . . 266
Initialization File . . . . 268
Batch Input to Produce
Examples . . . . . . . . . . 269
Accounting Management
Commands . . . . . . . . . . . . 270
Summary of Accounting
Management Command
Prototypes . . . . . . . . 271
ADD. . . . . . . . . . . 273
CONTINUE . . . . . . . . . 279
DELETE . . . . . . . . . . 280
DISPLAY . . . . . . . . . . 281

```
        EQUALIZE . . . . . . . . . }28
        EXPIRE . . . . . . . . . . }28
        HELP . . . . . . . . . . . }28
        MODIFY . . . . . . . . . . 284.1
        MTS, MCMD . . . . . . . . . }28
        RETURN . . . . . . . . . . }28
        SET . . . . . . . . . . . . }28
        STOP . . . . . . . . . . . }28
        SUBTRACT . . . . . . . . . }29
        Appendix A: Display Status
        Information . . . . . . . . . }29
        Appendix B: Multilevel
        Accounting . . . . . . . . . }29
Index . . . . . . . . . . . . . }29
```

THE SORT UTILITY PROGRAM

This section is designed to guide users of the Michigan Terminal System (MTS) in the use of the SORT utility program. It contains sufficient information to introduce a user to the relevant concepts and to allow him to utilize the program's various options. All descriptions and examples are couched in terms of the environment provided by MTS and users are expected to be cognizant of this environment.

SORT is a versatile program capable of ordering and structuring data sets. Sorting and merging facilities are provided to allow the user to arrange data according to a collating sequence he has described. Blocking and deblocking facilities, capable of processing data sets of types U, F, V, VS, FB, VB, VBS, and FBS, may be used alone or in conjunction with the sorting and merging options. Source (input) and sink (output) data sets may reside on any medium, e.g., cards, tapes, direct access devices, or virtual memory.

Subroutines, written by the user, can gain control from the SORT program at several points in the processing of the data: permitting the generation, modification, deletion, and comparison of the data. Users' subroutines may also rectify I/O errors and process program interrupts.

The SORT program may be invoked via the RUN command or may be called as a subroutine. The subroutines' linkage and idiosyncrasies are discussed in a latter portion of this guide.

There exists no limit to the volume of data that may be merged or copied; however, the available disk space as well as virtual memory space limits the volume of data that may be sorted. This limit is in excess of $50,000,000$ bytes at the University of Michigan, and additional facilities are provided to sort an amount of data up to the equivalent of a full magnetic tape.

The following descriptions may be useful:
blocking - a process of juxtaposing data records, forming a block of records to be written as a unit by an I/O device
byte - a unit of data consisting of eight contiguous bits - each EBCDIC character occupies one byte in the IBM 360, IBM 370, Amdahl 470, and Amdahl 5860 computers
collating field - contiguous bytes within each record whose attributes are used to order the records
collating sequence - a succession of records ordered by attributes of the data within each record
deblocking - a process of decomposing a group of contiguous data records read as a unit by an I/O device
halfword - a unit of data consisting of two consecutive bytes
merge - a process of arranging records from two or more previously sorted data sets to form a data set ordered on the same attributes as the source data
record - contiguous bytes that form a logical group of data to be processed as a unit by the SORT program
sort - a process of arranging records from one or more data sets to form a data set ordered on one or more attributes of the data
word - a unit of data consisting of four consecutive bytes

## SIMPLE CHARACTER SORTS

This section is designed to familiarize users with certain aspects of the control statement and to enable them to run simple sorts using the most common type of ordering: character, or alphabetized, collation. It is not intended to describe all, or even the best, ways to do simple character sorts. Rather, it will describe one way which produces the desired results. Complete descriptions of all aspects of the control statement, including the information presented in this section, appear later.

The sort program is invoked by entering a \$RUN *SORT command. The control statement, which describes the processing of the data, is placed in the PAR field of the \$RUN command. This control statement defines how the output is to be ordered and the locations of the input and output data. The following paragraphs describe a control statement for a simple character sort on quantities of data up to at least the lesser of 225,000 records and $18,000,000$ characters.

The first parameter to be specified in the PAR field is the keyword "SORT=" (or "S="). It is then necessary to specify the criteria for ordering the data. This is done by following "SORT=" with the positional parameters "CH,A,". The "CH" means that the ordering is to be based on character data, that is, data composed of alphanumeric characters or numbers in character form. Character data are produced, for example, by keypunches, terminals, and formatted output from FORTRAN. The "A" means that the output is to be in ascending order, e.g., the characters in the string " abzAZ019" are ascending. The \$RUN command should now look like this:

```
$RUN *SORT PAR=SORT=CH,A,
```

Note that a blank must not immediately precede any equal sign or comma in the control statement.

The location of the characters which are to determine the ordering of the records must now be supplied. Suppose that the field on which the sort is to be based starts in position, or column, 15 and ends in position 20 in each record. This is indicated to SORT as shown in the example below. The "A," in the control statement is followed by the starting position, "15" in this case, and the number of characters, or length, of the field, "6" in this case. The two values are separated by a comma. The \$RUN command is now

> \$RUN *SORT PAR=SORT=CH,A,15,6

If the designated fields (positions 15-20 in the example) of two records are compared and found to be identical, it may be desired to compare a second field in each record to determine which record should precede the other. This is accomplished by appending a comma and another CH,A,starting-position, number-of-characters combination to the
first description. Suppose that the second field occupies positions 79-80 of the records. The \$RUN command would then be
\$RUN *SORT PAR=SORT=CH, A, 15, 6, CH, A, 79, 2
Additional fields for the resolution of identical comparisons may be added in the same manner.

The next step is to specify where the input is to be found. This is done with the keyword "INPUT=" (or "I=") followed by a file or pseudodevice name, e.g., IN(5,99)@-TRIM or *I*. The \$RUN command might now be
\$RUN *SORT PAR=SORT=CH,A, 15,6, CH,A,79,2 INPUT=IN (5,99)@-TRIM
(In the case of blocked magnetic tapes, this is neither the most efficient nor the recommended method of completely specifying the input data set to SORT; but if tape blocking is left on, which is the default, it does work. It should also be noted that SORT does not reposition tapes either before or after using them nor does it write end-of-file marks.)

Finally, the place where $S O R T$ is to write the sorted output must be designated. The keyword "OUTPUT=" (or "O=") followed by a file or pseudodevice name accomplishes this. If the output is to be written to a tape with the pseudodevice name *O*, the completed \$RUN command would be

$$
\text { \$RUN *SORT PAR=SORT=CH,A,15,6,CH,A,79,2 I=IN }(5,99) @-T R I M O=* O *
$$

(The comments regarding magnetic input tapes also apply to tapes used for output.) The use of the same data set for both input and output should not even be considered unless the data would be easily recoverable if it were damaged and unless the user is knowledgeable about both SORT and the characteristics of the data set.

An alternate way of specifying the input and output data sets is to assign the input data set to SCARDS and the output data set to SPUNCH. Using this alternative, the \$RUN command for the same sort as above would look like:
\$R *SORT SCARDS=IN (5,99)@-TRIM SPUNCH=*O* PAR=S=CH,A,15,6,CH,A,79,2
When SORT finishes, it prints two statistics separated by a slash. The first of these is the number of records read by SORT. The second, which may normally be ignored, is the number of scratch files used by SORT.

## ELEMENTARY SORTING OF FORTRAN OUTPUT

Since FORTRAN is the most commonly used programming language, an introductory discussion of the sorting of data produced by programs written in FORTRAN is appropriate. If the input to SORT is the output from a FORTRAN program using formatted write statements, then the data are in character form and the control statement described in the previous section may be used. If the input to SORT was produced by unformatted write statements in a FORTRAN program, then it is generally necessary to modify the description of the fields to be compared; and additional information must be given to describe SORT's input and output.

In the preceding section, each field to be compared was described by a CH,A,starting-position,length combination. Since unformatted writes do not convert numeric data to character form, it is necessary to substitute the appropriate type code for the characters "CH". The following table indicates the type codes and lengths which should be used to sort FORTRAN variables.

| FORTRAN TYPE | CODE | LENGTH |
| :--- | :---: | :---: |
| INTEGER | FI | 4 |
| INTEGER*n | FI | n |
| REAL | FL | 4 |
| REAL*n | FL | n |
| DOUBLE PRECISION | FL | 8 |
| LOGICAL | BI |  |
| LOGICAL*n | $\mathrm{BI}^{1}$ | 4 |

Thus, to sort on an INTEGER*2 field starting in position 37 of each unformatted record, the $S O R T=$ portion of the control statement would be

SORT=FI, A, 37, 2
Additional fields may be added as indicated in the preceding section.
Because of the form in which FORTRAN programs write unformatted output, it is necessary to follow the input file or pseudodevice name with ",VS," which specifies the physical form of the records, and the maximum input record length. If the longest logical record in the file UNFORMAT is 300 bytes in length, then the INPUT= portion of the control statement would look like

INPUT=UNFORMAT,VS,300
1.TRUE. will follow .FALSE. in an ascending sequence.

```
Assuming that the output records from the sort are to be read by a FORTRAN unformatted read statement and, therefore, must have the same form as the input to SORT, it is necessary to follow the output file or pseudodevice name with a comma and the characters "VS".
Using the sort field and input file in the above examples and wanting to place the output in a file named OUT, the \$RUN command for SORT is
\$RUN *SORT PAR=SORT=FI,A,37,2 INPUT=UNFORMAT,VS,300 OUTPUT=OUT,VS
```

NOTATION

The following notation encodes the prototypes occurring in this section:

A uppercase letters denote parameters which must appear as printed
a lowercase letters denote generic parameters which must be replaced by an appropriate substitute
$=$, the equal sign and comma are used as stylistic delimiters in the prototypes; however, the equal sign and comma may be used interchangeably as delimiters

- denotes a single blank or space
... denotes zero or more repetitions of the preceding letters, words, or parameters
| denotes a choice of options, e.g., $x \mid y$ means choose $x$ or $y$
\{|\} denotes a choice of required alternatives, e.g., \{REC|MNR\} means either REC or MNR is required
[ ] denotes optional parameters, e.g., [structure] indicates that this parameter is optional
__ denotes a minimum abbreviation for a parameter, e.g., INPUT indicates that $I$ is the minimum abbreviation for input
$X^{\prime} y^{\prime}$ denotes the hexadecimal value $y$

THE CONTROL STATEMENT

The information needed by the SORT program to process the user's data is conveyed by a control statement. The control statement describes the mode of data processing, the collating sequence, if any, the structure of the source and sink data sets, the anticipated volume of the data, and the optional facilities to be utilized.

The control statement is read via GUSER if it is not passed as a parameter, i.e., "\$RUN *SORT PAR=control statement". If read, it may extend over any number of records and each record's length may range from 1 to 255 bytes.

Within any record of the control statement, consecutive keywords must be separated by at least one blank or space, e.g., "COPY.INPUT=...".

The control statement may be broken for continuation between parameters, immediately following a delimiter, except that a DS parameter must be completely contained in one record.

The control statement may be terminated by one of three methods: by an end-of-file sent via GUSER; by the END parameter; and by the end of the string of passed parameters.

Numeric values in the control statement must not exceed 15 digits, including leading zeros.

All parameters that are not defined in the control statement assume their default values.

A prototype of the control statement follows:
[COPY|[[SORT|MERGE] [=[[type], [aspect], [location], [length],]...
[type][,[aspect][,[location][,[length]]]]]]
[ [DS=delimiter[string]delimiter•]...DS=delimiter [string]delimiter]
[INPUT[=[[name], [structure], [record length], [block length],]...
[name] [, [structure] [, [record length] [, [block length]]]]]]
[OUTPUT[=[[name], [structure], [record length], [block length],]...
[name] [, [structure] [, [record length] [, [block length]]]]]
[additional parameter]...
[END]

The Mode's Description

The SORT program can operate in any of three modes: copying, sorting, or merging.

COPY indicates that the data should be copied from the source data set to the sink data set - being blocked or deblocked according to the specifications set forth in the source and sink data sets' descriptions.

SORT indicates that the data should be sorted. A collating sequence must be defined, if only implicitly. Deblocking and blocking of the source and sink data sets conforms to the user's specifications delineated in the descriptions of those data sets.

MERGE indicates that the source data sets should be merged. A collating sequence must be defined, perhaps implicitly. Deblocking and blocking of the source and sink data sets conforms to the user's specifications delineated in the descriptions of those data sets.

If no mode is specified by the user, SORT is assumed.

The Collating Sequence's Description

The SORT and MERGE modes require the definition of a collating sequence. The sequence is delineated through the description of collating fields. The order in which the collating fields are described (coded in the control statement) determines their precedence. The initial field is of primary importance, the second is of secondary importance, etc. The collating fields may overlap and may extend beyond the ends of records that vary in length, but cannot extend beyond the greatest implicit or declared record length.

The description of each collating field is formed from four parameters: the type of data contained in the field, the aspect of the field's sequence, the location of the field in a data record, and the length of the data within the field.

The relevant portion of the control statement's prototype is
[ [SORT|MERGE] [=[[type], [aspect], [location], [length], ] . . .
[type][,[aspect][,[location][,[length]]]]]]
For a single collating field, this may be simplified to
[SORT|MERGE] = type, aspect, location, length

## The following substitutions are necessary:

Replace "type" by the 1-character or 2-character code that describes the structure of the data contained in the field. Permissible codes are summarized below and are described in Appendix A. The default data type is character, CH.

| TYPE | CODE | SIGN PRESENT | FIELD LENGTH (BYTES) |
| :---: | :---: | :---: | :---: |
|  |  |  |  |
| alignment | AL | no | 1-4095 |
| binary | BI | no | $1-256$ |
| bit | BT | no | 1-255 (mask) |
| call | CA | - | 1-4095 |
| character | CH | no | 1-256 |
| defined sequence | DS(i) | no | 1-256 |
| fixed-point | FI | yes | 1-260 |
| floating-point | FL | yes | 2-16 |
| length | LE | - | - |
| packed decimal | PD | yes | 1-16 |
| sequence | SE | - | - |
| signed decimal | SD | yes | 1-17 |
| zoned decimal | ZD | yes | 1-16 |

Replace "i" by the single-character delimiter used with the DS keyword parameter which specifies the collating sequence for the defined sequence collating field.

Replace "aspect" by A or D denoting an ascending or descending collating sequence, respectively. The ascending sequence, A, is assumed if the aspect is not specified.

Replace "location" by the position in the record of the first byte, or character, of the collating field. The initial byte of a record is at location 1. If type V, VS, VB, or VBS records are processed, location 1 specifies the first byte of text in the record and not the record descriptor. The value of the location may range between 1 and 4092 with the exception of type FI fields with lengths greater than 4. In this latter case, the value of the location must be less than or equal to 4088. The default location is 1.

Replace "length" by the number of bytes of data in the collating field. Restrictions on the length of a field are dependent on its type and are noted in the table summarizing the type codes. The default length, regardless of the field type, is 80 bytes.

Examples of correct descriptions of the mode and collating sequence follow.

```
COPY indicates that the data should be copied
or C directly from the source data set to the
    sink data set
    indicates an ascending character sort on
    the first 80 bytes of each record
or S
or end-of-file
MERGE=CH,D,1,80 indicates a descending character merge on
or M=,D
or M,=D
SORT=SD,D,50,8,F,A,1,4,CH,A,10,1
or S,S,D,50,8,F,,,4,,,10,1 indicates a sort on the following fields
    of each record: a descending signed deci-
    mal field in the 50th through the 57th
    bytes, an ascending fixed-point field in
    the 1st through the 4th bytes, and an
    ascending character field in the 10th byte
SORT=AL,A, 8,4,CH,A, 8, 4, CA,A,1
or S=A,,8,4,,,8,4,CA indicates a sort on the following fields
or S=A,,8,4,,,8, of each record: an ascending alignment
4,CA
field in the 8th through the 11th bytes, a
    character field in the 8th through the
    11th bytes, and a field defined by the
    user via exit SORTE5
```

The Defined Sequences' Descriptions

Each different collating sequence for defined sequence collating fields must be specified in the control statement. These sequences are defined using the DS keyword.

The relevant portion of the control statement's prototype is
[[DS=delimiter[string]delimiter•]...DS=delimiter[string]delimiter]

This may be simplified to
[DS=delimiter string delimiter•]...
The following substitutions are necessary:
Replace "delimiter" by a single character (which may be a blank) which does not occur in "string" immediately preceding a blank. The delimiters for each DS keyword must be unique with respect to the delimiters used with any other DS keyword in the same control statement. The delimiter after "string" must be the same as the character before "string" and it must either be followed by a blank, be at the end of a record, or be at the end of the string of passed parameters.

Replace "string" by a string of characters representing an ascending collating sequence such that the first character in the string will precede the second character in the string, etc., in an ascending sort or merge. Any of the 256 bit combinations or characters may be used in the string. This includes the delimiter, provided that it is not immediately followed by a blank. If a character occurs in the string more than once, then all characters between the two occurrences of the repeated character will compare equal to the repeated character.

The sequence definition for a particular collating field is identified by placing the delimiter for the DS keyword in parentheses after the collating field type code DS in the collating sequence's description.

A DS keyword parameter may not be broken across records when the control statement is read via GUSER.

Examples of correct mode, collating sequence, and defined sequence descriptions follow.

```
S=DS (@),A,10,8,DS(@),A,20,8 DS=@0123456789ABCDEF@
or S=D(@),,10,8,D(@),,20,8 D=@0123456789@
    indicates a sort on two fields having
    character representations of hexadecimal
    digits; that is, the ascending sequence is
    "0123456789ABCDEF" - it is not necessary
    to include "ABCDEF" in the sequence defin-
    ition since any characters not included
    follow the defined sequence in their nor-
    mal character collating relationship to
    each other
```

S=D (:), , 10, $8, D(:), 20,8 \mathrm{D}=: 0123456789$ AaABbBCcCDdDEeEFfF:
indicates the same sort as above except
that the capital and lowercase representa-
tions of the letters may be used inter-
changeably so that capital and lowercase A
precede capital and lowercase $B$, which
precede capital and lowercase C, etc.

The Data Sets' Descriptions

Each source and sink data set to be used must be described in the control statement. Source data set descriptions should follow the INPUT keyword while sink data set descriptions should follow the OUTPUT keyword. Data sets are read in the order in which their descriptions appear unless the function is MERGE, in which case the data sets are processed simultaneously. Data sets are written in the order in which their descriptions appear, each being "filled" before the next is used.

Though data sets are not positioned, rewound, emptied, or terminated prior to or after processing, ${ }^{1}$ certain data sets, e.g., those residing in line files with only consecutive, integral line numbers, may be used both as the data's source and sink when sorting. However, this duplicate assignment of data sets is not generally recommended because of decreased efficiency and the possibility of the data being damaged if the sort is not completed.

The description of each data set is formed from four parameters: the data set's name, the structure of its records, the maximum length of its records, and the maximum length of its physical records or blocks. The characteristics, i.e., structure, record length, and block length, of all sink data sets must be alike.

The relevant portions of the control statement's prototype are
[INPUT[=[[name], [structure], [record length], [block length],]... [name] [, [structure] [, [record length] [, [block length]]]]]]
[OUTPUT[=[[name], [structure], [record length], [block length],]... [name] [, [structure] [, [record length] [, [block length]]]]]]

For single source and sink data sets, these may be simplified to

```
INPUT=name,structure,record length,block length
```

OUTPUT=name,structure, record length,block length

The following substitutions are necessary:
Replace "name" by the file or pseudodevice name of the data set. Modifiers and line number ranges may be appended to the name. If the name of the source data set is omitted, data will be read from the data set assigned to SCARDS, e.g., "\$RUN *SORT SCARDS=source". If the name of the sink data set is omitted, data will be written into the data set assigned to SPUNCH, e.g., "\$RUN *SORT SPUNCH=sink".

Replace "structure" by one of the record structure codes appearing in the table below. ${ }^{2}$ If the structure's code is omitted, the "U" structure is assumed for source data sets and for sink data sets unless a structure code is specified for another sink data set, in which case the specified structure is assumed. Appendix $B$ and OS/VS2 MVS Data Management Services Guide, IBM publication GC26-3875, contain descriptions of the structures.
${ }^{1}$ See, however, the description of the TPS parameter in the section entitled "Additional Parameters."
${ }^{2}$ See also the section entitled "Processing Magnetic Tapes."

| CODE | RECORD STRUCTURE |
| :--- | :--- |
| U | undefined length |
| F | fixed length |
| V | variable length |
| VS | variable length; spanned |
| FB | fixed length; blocked |
| VB | variable length; blocked |
| VBS | variable length; blocked; spanned |
| FBS | fixed length; blocked; standard |

Replace "record length" by the byte length of the longest record occurring in the data set. The record length must be greater than 0 and less than 32760 . If the record length is not specified, a default length is assumed as described below.

Replace "block length" by the byte length of the longest block occurring in the data set. The block length must be greater than 0 and less than 32768; for type FB blocks, it must be less than 32764. The block length defaults as described below.

Computation of default record and block lengths involves the maximum data set block length (MDSBL). This is defined for each input data set as the maximum input length for the data set, which is returned by the subroutine GDINFO. ${ }^{1}$ The maximum input length returned by GDINFO is sufficiently large enough for all blocks which may be read from the data set. The MDSBL for each output data set is defined as the smallest of all the maximum output lengths which are returned by the subroutine GDINFO for the output data sets. This is the length of the longest block which may be written into all the output data sets without truncation. If GDINFO cannot return a length, such as for data sets which are not specified or do not exist at the time of the invocation of SORT, then the maximum length is assumed to be 80. The MDSBL is always based on the first element of an explicit or implicit concatenation; and hence may not produce the desired default in the case of concatenated data sets.

The default record and block lengths are indicated in the following table.

[^0]| Record Structure |  | U F | $\begin{aligned} & \text { FB } \\ & \text { FBS } \end{aligned}$ | V | VB | VS | VBS |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| In | Record | max (MDSBL, 80) |  | max (MDSBL-8, 80) |  |  |  |
|  | Length |  |  |  |  |  |  |
|  | Block |  | MDSBL - 1 + RL - rem | RL | max (RL+8, | min (RL+8, |  |
|  | Length | RL | ( (MDSBL-1) /RL) | +8 | MDSBL) | MDSBL) | MDSBL |
| Out* | Record | largest input record length |  |  |  |  |  |
|  | Length |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  | Block |  | max (RL, MDSBL- | RL | max (RL+8, | min (RL+8, |  |
|  | Length | RL | rem(MDSBL/RL)) | +8 | MDSBL) | MDSBL) | MDSBL |

MDSBL = maximum input length returned by GDINFO for input; smallest maximum output length returned by GDINFO for output; (80 assumed for undefined maximum lengths)

RL = source data set record length for input; sink data set record length for output

MDSBL-1+RL-rem((MDSBL-1)/RL) = the smallest integral multiple of the data set record length which is greater than or equal to the MDSBL

MDSBL-rem (MDSBL/RL) $=$ the largest integral multiple of the data set record length which is less than or equal to the MDSBL

* If the record or block length is specified for any sink data set, then that length is used as the respective default record or block length for all sink data sets.

The default input record length may be greater than the length of the longest actual record in the data set. In this case, specifying a record length equal to the length of the longest record will produce greater efficiency. The default block lengths are defined so as to provide maximum blocking for the record structure, record length, and data set being used.

The user should be aware of the implicit "trimming" of blanks in MTS. The SORT program does not set the $\neg T R I M$ modifier when reading or writing data sets.

Examples of correct data set descriptions follow.

INPUT=FILE1, U, 80, 80
or I=FILE1,,80
indicates that the data should be read from a file named FILE1 - the records are of structure $U$ and have a maximum length of 80 bytes

OUTPUT=OUT, F, 255, 255
or O,OUT,F,255

INPUT=X, U, 80, 80, Y, U, 80, 80
or $\mathrm{I}=\mathrm{X}, \mathrm{8} 0, \mathrm{Y}, 80$
indicates that the data should be written into a file named OUT - the records are of structure $F$ and all have a length of 255 bytes
indicates that the data should be read from files $X$ and $Y$ - both files' records are of structure $U$ with a maximum length of 80 bytes

INPUT=FILE1 (1, 100) @ ᄀTRIM, U, 80, 80,FILE2,FB, 40, 400
or I,FILE1 (1,100)@ TTRIM, 80, ,FILE2,FB, 40,400
indicates that the data should be read from two files - FILE1's records are of structure $U$ and have a maximum length of 80 bytes; FILE2's records are of structure FB and have a record length of 40 bytes and a block length of 400 bytes

OUTPUT=*Y*,VB, 800,32767 INPUT=*X*, U, 800,800
or $\mathrm{O}=* \mathrm{Y} *, \mathrm{VB}, 32767 \mathrm{I}=* \mathrm{X} *,, 800$
or $I=* X *, 800$ indicates that the data should be read $O=* Y *, V B, 32767$ from a pseudodevice named *X* and written onto a pseudodevice named *Y* - the source records are of structure $U$ with a maximum length of 800 bytes - these records are to be restructured to type VB output records with a maximum record length of 800 bytes and a maximum block length of 32,767 bytes

## The END Parameter

The END parameter may be used to terminate the control statement. The relevant portion of the control statement's prototype is
[END]
Users of the SORT subroutine must use this parameter to indicate the end of the control statement.

## Additional Parameters

Any or all of the following parameters may appear in the control statement.

The relevant portion of the control statement's prototype is
[additional parameter]...

## CHK (exit check facility)

The check facility may be used to assist one in testing the subroutines used in exits SORTE1 through SORTE5. It should be used only when processing a limited quantity of data.

When the check facility is active, every call from an enabled exit to a user's subroutine is noted on SERCOM as is the subroutine's return to SORT. Relevant parameters are checked for accuracy and passed to SERCOM for display.

DEC (delete comments)
The DEC parameter may be included to suppress the printing of the statistics and diagnostics that are normally generated by the SORT program. Diagnostics resulting from errors detected while processing the control statement will not be suppressed unless the DEC parameter is processed before the error is detected.

DEL=x[,x]... (delete output records)
The DEL parameter may be used with either the SORT or MERGE modes to specify that certain classes of output records are to be discarded before writing the sink data set. This deletion is based on whether or not two consecutive output records are duplicates, that is, whether or not they are equivalent according to the description of the collating sequence. For the purpose of the DEL parameter, two records are always considered to be duplicates whenever it is necessary to use the SE collating type to differentiate between them.

Replace "x" by one of the parameter values in the following table.

| VALUE | RECORDS DELETED FROM EACH GROUP OF DUPLICATES |
| :--- | :--- |
| DUP | all |
| FIRST | the first |
| LAST | the last |
| LEAD | all except the last |
| SINGLE | none; all nonduplicate records are deleted |
| TRAIL | all except the first |

Multiple specification of DEL parameter values produces cumulative effects; for example, DEL=FIRST,TRAIL is the same as DEL=DUP.

## HELP [topic]

The HELP parameter provides on-line assistance for the use of the program. If no parameter is specified, general assistance is provided; otherwise, assistance on a specific "topic" is provided. The remainder of the line is taken as the topic.

May 1983

LIO (list data set characteristics)
The LIO parameter may be included to produce a listing on SERCOM of the record structure, record length, block length, and name, if known, for each data set.
\{ㄹEC|MNR \} =x (number of records)
The efficient allocation of resources during a sort necessitates that an estimate of the number of data records be included in the control statement. Though the estimate may be omitted, its inclusion is strongly recommended.

Replace "x" by an estimate of the maximum number of records contained in the source data set(s). The estimate of the number of records must be greater than 0 and less than 2147483648. If no estimate is included, a quantity of 5000 records is assumed.

Note that the inclusion of this parameter does not limit the number of records read.

RES=x (restart)
The restart facility may be used to restart a sort that terminated prematurely due to an error detected while writing a data set or that resulted in faulty output due to the characteristics of the sink data set. It may be used if the following criteria are met:
i. the error occurred during a terminal session
ii. the terminal session is still active
iii. a warning or error message was issued
iv. the sort indicated that one or more intermediate files were used to process the data (see the section entitled "Statistics and Diagnostics")
v. no sort creating or using intermediate files has been invoked during the interim
vi. the TPS parameter was not used
vii. the source of the error has been corrected, e.g., replacing a faulty tape, creating a larger output data set, or respecifying the blocking or length parameters to the MTS tape routines

The control statement for the restarted sort should be identical to that of the initial sort save for the addition of RES=x, "x" being the number of intermediate files that the initial sort used.

## (sign off on error)

The SIG parameter may be used to produce a signoff if SORT detects an error. An error detected while processing the control statement will produce a signoff only if the SIG parameter is processed before the error is detected.

## TPS $[=\{x \mid$ name, name[, name]...\}] (tape-merge sort facility)

Disk files are normally used for intermediate data sets. However, if there is insufficient disk space or virtual memory space for the volume of data to be sorted, the TPS parameter may be used to designate that the intermediate data sets are to be tapes. This facility should be used only in the case of the above conditions since the merge portion of the sort is considerably less efficient than in the case of intermediate disk files.

Replace "x" with the number of tapes to be used for the merge phase of the sort. If "x" is omitted, the minimum value, 3, is assumed. A value of "x" greater than 3 increases the efficiency of the merge. For $x=4$, the theoretical decrease in merge time over $x=3$ is approximately $33 \%$. For $x=5$, the theoretical reduction over $x=4$ is approximately $15 \%$. For larger values of "x", the percentage decrease for $x+1$ over $x$ is only a little over half the reduction for $x$ over $x-1$. The merge order is $x-1$ and the total number of tape drives required for the sort is i+o-1+x, where "i" is the number of source data sets which are tapes and "O" is the number of sink data sets which are tapes. The $x-1$ intermediate tapes are mounted and dismounted automatically.

As an alternative, replace "name" with the pseudodevice name of a tape which may be used for an intermediate data set. With this option, at least two tapes must be specified. The merge order is the same as the number of specified tapes; i.e., specifying three tapes is equivalent to specifying $x=4$. The enablement of blocking, the block size, the expiration date, and the mode for each of the tapes specified in this manner may be changed and will not be restored. If "name" is omitted, an intermediate tape will be mounted and dismounted automatically.

The first sink data set must be a tape, and it is assumed that this tape may be used for intermediate results. Consequently, the sorted output will be written from the beginning of the first data set. Generally, the first sink data set cannot also be a source data set.

The maximum volume of data which may be sorted at one time using a tape-merge is approximately the capacity of one maximum density tape. If more than this amount is to be sorted, each tape should be sorted separately and the sorted output should then be merged.

If the TPS parameter is used, the MERGE and COPY keywords should not appear in the control statement. The data type $S E$ and the RES parameter cannot be used. The parameters REC, MNR, and PGS are ignored.

## Restricted Parameters

The parameters described in this section are additional and have limited application. They should not be included in the control statement without consulting a member of the Computing Center staff.

ARL=x (average record length)
If the average record length is significantly less than the maximum record length, this parameter may reduce the number of intermediate files required by a sort. If "x" is omitted, a length of one-half the maximum record length will be assumed. It is suggested that the use of this parameter be considered only if "x" is less than one-half the maximum record length, as its use may actually increase the number of intermediate files.

DIO (I/O error trap)
This parameter disables the I/O error trap normally set by the SORT program.

DPI (program interrupt trap)
This parameter disables the program interrupt trap normally set by the SORT program.

MBY=x (number of bytes)
This parameter limits the amount of storage that may be used for an internal sort. It is suggested that "x" be greater than $10^{4}$ bytes and less than $10^{6}$ bytes.

PGS $=x$ (pages)
This parameter limits the anticipated maximum size and the number of pages requested whenever an intermediate file is created by the SORT program. It is suggested that "x" be greater than 25 pages and less than 500 pages.

OPTIONAL EXITS

The SORT program can pass control to users' subroutines at any of ten points or exits. These exits permit the user to mold the SORT program to his requirements: his subroutines could generate, modify, delete, or compare records, or could recover from I/O errors or program interrupts.

An exit may exist in either of two states, enabled or disabled. Whenever an exit is enabled, the SORT program relinquishes control to the appropriate subroutine during the data's processing. Whenever an
exit is disabled, the SORT program retains control during the data's processing. All exits are assumed disabled unless explicitly enabled.

There are two methods to enable an exit and one method to dynamically disable an exit. An exit is enabled if the exit's name, e.g., SORTE2 or SORTE4, is encountered as an external symbol during the loading of the SORT program: thus the user's subroutine should bear the name of the exit it is to enable and should be loaded along with the SORT program. An exit is enabled or disabled if the entrance SORTEA is called and given appropriate parameters.

All subroutine calls are made via a standard OS linkage, type (I) S. Upon entry, general register 1 will point to a parameter list, a vector of address constants, register 13 will point to a "save area", register 14 will contain the return address, and register 15 will contain the subroutine's entry point. MTS Volume 3, System Subroutine Descriptions, contains an explanation of the type (I) S linkage.

All subroutines should return control to the SORT program via the standard OS (I) $S$ return. Upon return, the contents of general registers 2 through 13 must be identical to their contents when the subroutine was called. The significance of a subroutine's return code, the contents of general register 15 when the subroutine returns to the SORT program, is discussed in the descriptions of the exits. FORTRAN subroutines produce a return code of "n" by using a "RETURN i" statement, where "i" has the value n/4.

Exit subroutines may change the I/O error trap exit (SETIOERR or CUINFO subroutines), the program interrupt trap exit (PGNTTRP, SPIE, or CUINFO subroutines), or the program mask (SPIE subroutine or SPM machine instruction). Before returning to SORT, if any of these have been changed, they must be restored to the condition at the time the exit subroutine was called. ${ }^{1}$ Normally the SORT program processes all I/O errors and program interrupts; however, the exits SORTE6, SORTE7, and SORTE8 permit the user to acquire control in such circumstances.

Exit SORTEO

If this exit is enabled, it is taken immediately after the SORT program gains control from the operating system or its calling program. The user's subroutine is called before the control statement is decomposed, hence prior to the processing of the data. The subroutine could enable other exits, initialize parameters, or initialize data sets.
${ }^{1}$ See the descriptions of the GUINFO and SPIE subroutines in MTS Volume 3, System Subroutine Descriptions, and the descriptions of the BALR and BAL machine instructions.


#### Abstract

The sole entry in the parameter list points to the text passed to the SORT program by its calling program or the operating system. If the SORT program is invoked by the RUN command, the text is preceded by a halfword containing the length of the text. If the SORT program is called as a subroutine, the parameter points directly to the text.

The subroutine's return code is significant. A return code of 0 indicates that SORT should initiate processing of the data while a return code greater than 3 indicates that SORT should terminate processing. In the second case, a return code of 4 is passed to the program that called SORT or to the operating system.

The following FORTRAN subroutine exemplifies the use of the exit SORTEO. It rewinds logical I/O units 5 and 7.


SUBROUTINE SORTEO
REWIND 5
REWIND 7
RETURN
END

Exit SORTE1

If this exit is enabled, it is taken to acquire a block (or unblocked record) for processing. Calls to the user's subroutine replace the calls that would normally be made to the READ subroutine.

A block may be moved into the region addressed by the first word of the parameter list, and the length of the block should be placed in the halfword addressed by the second word of the parameter list. The fifth word of the parameter list points to either the FDUB or the eightcharacter, left-justified, logical unit name of the current source data set specified in the control statement. Both the region and length are aligned on halfword boundaries. A block whose returned length is zero or less is discarded, i.e., ignored: no error results and SORTE1 is called again for another block.

If this exit is enabled, a dummy data set, e.g., named *DUMMY*, must be specified for each source data set defined in the control statement. If the data sets are to be merged, the specified data set names should be either the actual names of the data sets or other nondefault names so that the FDUB ${ }^{1}$ pointed to by the fifth word of the parameter list may be utilized to determine the data set from which a block is to be acquired. The data set's parameters should describe the records to be passed to the SORT program by the user's subroutine. The deblocking facilities may be used to decompose blocks passed by the subroutine.
${ }^{1}$ See the descriptions of the READ and GDINFO subroutines in MTS Volume 3 , System Subroutine Descriptions.

The return codes passed from the user's subroutine to the SORT program are pertinent. A code of 0 indicates that a block was returned and that processing should continue. A code of 4 indicates that no block was returned, that this data set has been exhausted, and that processing should continue with the next source data set to be defined in the control statement. If another data set does not exist, processing continues to its logical conclusion. A return code greater than 7 indicates that processing should be discontinued. In this case, SORT returns a code of 4 to its calling program or produces a signoff if the control statement contains the SIG parameter.

The subroutine appearing on the next page and the eighth example in the section entitled "Sort Examples" exemplify the use of exit SORTE1.

The following FORTRAN subroutine and the one which appears in the section describing the exit SORTE4 illustrate one method to maintain the continuity of consecutive data records that are to be processed as a group.

For example, assume that information pertaining to a single family has been coded on two 80 -byte cards. The first record of each pair contains information specific to the family, e.g., number of children, while the second contains information about their dwelling. A list of the data, ordered by number of children, is desired; however, the original pairing must be retained.

The following FORTRAN subroutine combines, blocks, two 80-byte records to form a single record to be processed by the SORT program.

The data set's description could be "I=*DUMMY*,F,160,160".
A companion subroutine, SORTE4, should be used in conjunction with this subroutine. It will separate the juxtaposed records.

```
                SUBROUTINE SORTE1(REGION,LENGTH,*,*)
                LOGICAL*1 REGION(160)
                INTEGER*2 LENGTH
                READ (5,END=4) (REGION (I),I=1, 80)
                READ (5,END=8) (REGION (I),I=81,160)
                LENGTH=160
                RETURN
END
```

4 RETURN 1
8 RETURN 2

Exit SORTE2

If this exit is enabled, it is taken immediately after a record has been acquired for processing, following deblocking, if any. The user's subroutine may delete or modify the record before it is subsequently processed.

The first word of the parameter list will point to the record; the second, to its halfword length plus 4. The record and its length are aligned on halfword boundaries.

If a record's length is altered by the subroutine, the new length plus 4, must replace that passed to the subroutine. If the new record length is greater than the original length, a record length at least as large must have been specified for at least one of the input or output data sets. For structure $\mathrm{F}, \mathrm{FB}$, and FBS records, this may mean that a dummy input data set must be specified, e.g., *DUMMY*,U,new-recordlength. If a length of zero is returned, the record is discarded: this implies that the length must be set to 0 and not 4.

The return codes passed from the user's subroutine to the SORT program are significant. A code of 0 indicates that a record was returned and that processing should continue. A code of 4 indicates that no record was returned and that processing should continue with the next source data set to be defined in the control statement. If another data set does not exist, processing continues to its logical conclusion. A return code greater than 7 indicates that processing should terminate. In this case, SORT returns a code of 4 to its calling program or produces a signoff if the control statement contains the SIG parameter.

The following FORTRAN subroutine exemplifies the use of the exit SORTE2. It deletes records having blanks in columns 3 and 4, terminates the reading of a data set if columns 3 and 4 contain '09', and returns an error code of 8 if columns 5 and 6 contain '09'.

```
SUBROUTINE SORTE2(REGION,LENGTH,*,*)
INTEGER*2 REGION(40),BLANK/' '/,NINE/'09'/,LENGTH
IF (REGION(2).EQ.BLANK) GO TO 100
IF (REGION(2).EQ.NINE) GO TO 200
IF (REGION(3).EQ.NINE) GO TO 300
RETURN
100 PRINT 900, (REGION(I), I=1,40)
900 FORMAT(' RECORD DISCARDED:',40A2)
LENGTH = 0
RETURN
200 RETURN 1
300 RETURN 2
END
```

The following example and the corresponding one appearing in the section describing SORTE3 illustrate one method of redefining the normal
character collating sequence. A translation scheme similar to this is usually more efficient than using a defined sequence (DS) collating field or performing the collation through exit SORTE5.

Assume that the following collating sequence were desired:

> - ф.<(+|\&!\$*); - - , \%_>? :\#@' ="0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ

Note that this sequence is identical to the existing character sequence except that the digits precede the letters. Further assume that only these graphics will appear in the data, e.g., that $\mathrm{X}^{\prime} \mathrm{BO}^{\prime}$, X'B1',..., X'B9' will not appear.

The following subroutine translates all the digits appearing in the data, $X^{\prime} F 0^{\prime}, X^{\prime} F 1^{\prime}, . . . X^{\prime} F 9^{\prime}, ~ i n t o ~ t h e ~ c h a r a c t e r s ~ X^{\prime} B 0^{\prime}, X^{\prime} B 1^{\prime}, \ldots . X^{\prime} 9^{\prime}$, respectively. An indication of how to write the equivalent subroutine in FORTRAN is shown in the examples of the section describing SORTE3. When used in conjunction with SORTE3's example and a character collating sequence, e.g., "C,A,5,17", it will produce the desired sequence.

| SORTE2 | CSECT |  |  |
| :---: | :---: | :---: | :---: |
|  | USING | *,R15 |  |
|  | LM | R0,R1, 0 (R1) | **LOAD PARAMETERS |
|  | LH | R1, 0 (0,R1) | LOAD RECORD LENGTH |
|  | S | R1, =F'5' | DECREMENT RECORD LENGTH |
|  | BM | RTRN | LENGTH < 0 |
|  | STC | R1, *+7 | SET LENGTH IN "TR" |
|  | LR | R1, R0 |  |
|  | TR | $0(0, R 1)$, TTAB | TRANSLATE |
| RTRN | SR | R15,R15 | RETURN CODE |
|  | BR | R14 | RETURN |
| TTAB | DC | XL (4*16)'00' |  |
|  | DC | X'404142434445 | 48494A4B4C4D4E4F' |
|  | DC | X'50515253545 | 58595A5B5C5D5E5F' |
|  | DC | X'606162636465 | 68696 66B6C6D6E6F' |
|  | DC | X'70717273747 | 78797A7B7C7D7E7F' |
|  | DC | XL (4*16)'00' |  |
|  | DC | X' C 0 C 1 C 2 C 3 C 4 | C8C9 CACBCCCDCECF' |
|  | DC | X'D0D1D2D3D4D | D8D9DADBDCDDDEDF' |
|  | DC | $X^{\prime} E 0 D 1 E 2 E 3 E 4$ | E8E9EAEBECEDEEEF' |
|  | DC | $X^{\prime}$ B0B1B2B3B4 | 38B9FAFBFCFDFEFF' |
| **EQUIVALENCES |  |  |  |
| R0 | EQU | 0 |  |
| R1 | EQU | 1 |  |
| R14 | EQU | 14 |  |
| R15 | EQU | 15 |  |
|  | END |  |  |

Exit SORTE3

If this exit is enabled, it is taken after a record has been processed and immediately prior to blocking, if any. The user's subroutine may delete or modify the record before it is subsequently written.

The first word of the parameter list will point to the record; the second, to its halfword length plus 4. The record and its length are aligned on halfword boundaries.

If a record's length is altered by the subroutine, the new length, plus 4, must replace that passed to the subroutine. If the new record length is greater than the original length, a record length at least as large must have been specified for at least one of the input or output data sets. For structure $F, F B$, and $F B S$ records, this may mean that a dummy input data set must be specified, e.g., *DUMMY*,U,new-recordlength. If a length of zero is returned, the record is discarded: this implies that the length must be set to 0 and not 4.

A return code of 0 indicates that processing should continue. A return code of 4 indicates that processing should continue, that the current sink data set should be closed, and that the next data set to be defined in the control statement, if any, should be opened and filled. If another data set is not defined in the control statement, processing terminates with an error comment. A return code greater than 7 indicates that an error occurred and that processing should terminate immediately. In this case, a return code of 4 is passed to SORT's calling program or to the operating system unless the control statement contains the SIG parameter, in which case a signoff is produced.

The following FORTRAN subroutine deletes the second and subsequent occurrences of identical records.

```
SUBROUTINE SORTE3(R1,L1)
INTEGER*2 R1(128),R2(128),L1,L2/0/
LX = L1 - 4
IF (L1.NE.L2) GO TO 2
I = ICLC(LX,R1,0,R2,0)
IF (I.NE.O) GO TO 4
L1 = 0
RETURN
4 CALL IMVC(LX,R2,0,R1,0)
    RETURN
END
```

2 L2 = L1

The following subroutine is a companion to that discussed in the section describing the exit SORTE2. It translates $\mathrm{X}^{\prime} \mathrm{B} 0^{\prime}, \mathrm{X}^{\prime} \mathrm{B} 1^{\prime}, \ldots . . \mathrm{X}^{\prime} \mathrm{B} 9^{\prime}$ to the digits $X^{\prime} F 0^{\prime}, X^{\prime} F 1^{\prime}, . . . X^{\prime} F 9^{\prime}$.

| SORTE3 | CSECT |  |  |
| :---: | :---: | :---: | :---: |
|  | USING | *, R15 |  |
|  | LM | R0,R1, 0 (R1) | **LOAD PARAMETERS |
|  | LH | R1, 0 (0,R1) | LOAD RECORD LENGTH |
|  | S | R1, =F'5' | DECREMENT RECORD LENGTH |
|  | BM | RTRN | LENGTH < 0 |
|  | STC | R1, *+7 | SET LENGTH IN "TR" |
|  | LR | R1, R0 |  |
|  | TR | 0 (0,R1) , TTAB | TRANSLATE |
| RTRN | SR | R15, R15 | RETURN CODE |
|  | BR | R14 | RETURN |
| TTAB | DC | XL (4*16)'00' |  |
|  | DC | X'40414243444 | 48494A4B4C4D4E4F' |
|  | DC | X'50515253545 | 58595A5B5C5D5E5F' |
|  | DC | X'6061626364 | 6869 6A6B6C6D6E6F' |
|  | DC | X'70717273747 | 78797A7B7C7D7E7F' |
|  | DC | XL (3*16)'00' |  |
|  | DC | X'F0F1F2F3F4F | F8F9BABBBCBDBEBF' |
|  | DC | $\mathrm{X}^{\prime} \mathrm{C} 0 \mathrm{C} 1 \mathrm{C} 2 \mathrm{C} 3 \mathrm{C} 4$ | C8C9 ${ }^{\text {cacbeccicecF }}$ |
|  | DC | X'D0D1D2D3D4D | 88D9DADBDCDDDEDF' |
|  | DC | X'E0D1E2E3E4 | E8E9EAEBECEDEEEF' |
|  | DC | $X^{\prime}$ F0F1F2F3F4F | F8F9FAFBFCFDFEFF' |
| **EQUIVALENCES |  |  |  |
| R0 | EQU | 0 |  |
| R1 | EQU | 1 |  |
| R14 | EQU | 14 |  |
| R15 | EQU | 15 |  |
|  | END |  |  |

The following program is the equivalent FORTRAN subroutine.

```
SUBROUTINE SORTE3(/REGION/,LENGTH)
INTEGER*2 LENGTH
REAL TRANS(3)/ZB0B1B2B3,zB4B5B6B7,zB8B90000/
CALL TRNC(LENGTH - 4,REGION,TRANS,'0123456789',10)
RETURN
END
```

Exit SORTE4

If this exit is enabled, it is taken to dispose of a processed block (or unblocked record). Calls to the user's subroutine replace the calls that would normally be made to the WRITE subroutine.

The first word of the parameter list will contain the address of the current output block, and the second word will contain the address of the halfword length of the block. The fifth word of the parameter list points to either the FDUB or the eight-character, left-justified, logical unit name of the current sink data set specified in the control
statement. Both the block and its length are aligned on halfword boundaries.

If this exit is enabled, a dummy data set, e.g., named *DUMMY*, must be specified for each sink data set defined in the control statement. The data set's parameters should describe the records to be passed to the user's subroutine. The blocking facilities may be used.

The subroutine's return code is immaterial.
The following FORTRAN subroutine is a companion to that discussed in the section describing the exit SORTE1. It decomposes a 160-byte record into two 80-byte records.

The data set's description could be "O=*DUMMY*,F,160,160".

```
SUBROUTINE SORTE4(REGION)
LOGICAL*1 REGION(160)
WRITE (7) (REGION(I),I=1,80)
WRITE (7) (REGION(I),I=81,160)
RETURN
END
```

Exit SORTE5

If this exit is enabled, it is taken to compare two records. Calls to the user's subroutine replace the comparisons normally made by the SORT program. The user's subroutine is expected to return the result of the comparison to SORT.

The user's subroutine is entered whenever the CALL option is encountered in the description of the collating sequence. Were the mode definition $" S=C A, A, 1,4 "$, the user's subroutine would be entered every time two records were to be compared. Were the mode definition "S=S,D, 12, 4, CA, D, 1, 2, F, A, 59, 4, CA, A, 4, 6", the user's subroutine would be entered once, for every two records being compared, if the first field, "S,D,12,4", failed to order the records and once if both the first CALL and subsequent "F,A,59,4" fields failed to order the records. In this case, the subroutine may be called from zero to two times during the comparison of any two records.

The parameter list passed to the user's subroutine differs from that of the preceding exits. The first word of the parameter list points to a two-word array with the first word containing the "location" and the second word containing the "length" coded in the collating field's description. The second and third words of the list point to two halfword-aligned regions containing the records for comparison. The first halfword of each region contains a value of 4 plus the length of the text in that region. The text begins at an address 4 bytes higher than that of the region.

The subroutine's return code indicates the result of the records' comparison. The interpretation of the code depends on the aspect specified in the definition of the CALL collating field.

If an ascending collating sequence, $A$, is specified, the return codes have the following meaning. A return code of 8 indicates that the record appearing in the first region, i.e., the region addressed by the second parameter, should precede the record appearing in the second region. A return code of 4 indicates that either record may precede the other. A return code of 0 indicates that the second region's record should precede the first region's record.

If a descending sequence, $D$, is specified, the return codes have the following meaning: a code of 0 indicates that the first region's record should precede the second region's record, a code of 4 indicates that either record may precede the other, and a code of 8 indicates that the second region's record should precede the first region's record.

The following subroutine compares the lengths of the two records passed as parameters. The record with the longer length is defined to be the predecessor if an ascending collating sequence is specified, e.g., "CA,A".

| SORTE5 | CSECT |  | **LOAD PARAMETERS |  |
| :---: | :---: | :---: | :---: | :---: |
|  | L | R15, 8 (0,R1) |  |  |
|  | L | R1, 4 (0,R1) |  |  |
|  | CLC | 0 (2,R15) , 0 (R1) | COMPARE | RECORD LENGTHS |
|  | LA | R15,8 | RETURN | CODE $=8$ |
|  | BHR | R14 |  |  |
|  | LA | R15,4 | RETURN | CODE $=4$ |
|  | BER | R14 |  |  |
|  | SR | R15, R15 | RETURN | CODE $=0$ |
|  | BR | R14 |  |  |
| **EQUIVALENCES |  |  |  |  |
| R1 | EQU | 1 |  |  |
| R14 | EQU | 14 |  |  |
| R15 | EQU | 15 |  |  |
| END |  |  |  |  |

The following FORTRAN program is functionally the same as the above subroutine.

```
SUBROUTINE SORTE5(LOC,REC1,REC2,*,*)
INTEGER*2 REC1,REC2
IF (REC1 - REC2) 10,20,30
RETURN
RETURN 1
RETURN 2
END
```

In ordering the records according to the character collating sequence, the following FORTRAN subroutine ignores the first character in the collating field if that character is an asterisk.

```
SUBROUTINE SORTE5(FIELD,REC1,REC2,*,*)
INTEGER FIELD(2),SW
LOGICAL*1 REC1(1),REC2 (1),EQUC*4
LOC=FIELD (1) +4
SW=0
IF (EQUC(REC1 (LOC),'*')) SW=1
IF (EQUC(REC2(LOC),'*')) SW=SW-1
SW=1-(LCOMC (FIELD (2) - IABS (SW), REC1 (LOC+ (1+SW)/2),
C REC2(LOC+(1-SW)/2)))
RETURN SW
END
```

Exit SORTE6

If this exit is enabled, it is taken whenever an error is detected while acquiring a record for processing, i.e., through a call to the READ subroutine. Normally the SORT program terminates in such circumstances; however, if this exit is enabled, the user's subroutine may attempt to rectify the error and resume processing.

The parameter list is identical to that passed to READ: the first word of the list points to a region to receive the next record, the second word points to a halfword to receive the next record's length, the third word points to a word containing modifiers, the fourth word is irrelevant, and the last word of the parameter list points to either the FDUB or the eight-character, left-justified, logical unit name of the current source data set.

The subroutine's return codes are pertinent. A return code of 0 indicates that processing should continue with the current source data set. A return code of 4 indicates that processing of the current source data set should terminate and resume with the next source data set to be defined in the control statement, if any. In either event, any record which was actually acquired when the error occurred will be ignored. A return code greater than 7 indicates that processing should terminate because of the error. In this case, a return code of 4 is passed to SORT's calling program or to the operating system unless the control statement contains the SIG parameter, in which case a signoff is produced.

Exit SORTE7

If this exit is enabled, it is taken whenever an error is encountered while disposing of a processed record, i.e., through calls to the WRITE subroutine. Normally the SORT program terminates in such circumstances; however, if this exit is enabled, the user's subroutine may attempt to rectify the error and resume processing.

The parameter list is identical to that passed to WRITE: the first word of the list points to a record to be written, the second word points to a halfword containing the record's length, the third word points to a word containing modifiers, the fourth word is irrelevant, and the last word of the parameter list points to either the FDUB or the eight-character, left-justified, logical unit name of the current sink data set.

The subroutine's return codes are relevant. A return code of 0 indicates that processing should continue with the current sink data set. A return code of 4 indicates that processing of the current sink data set should terminate and resume with the next sink data set to be defined in the control statement, if any. In either event, the record which was being disposed of when the error occurred will be reprocessed. A return code greater than 7 indicates that processing should terminate because of the error. In this case, a return code of 4 is passed to SORT's calling program or to the operating system unless the control statement contains the SIG parameter, in which case a signoff is produced.

Exit SORTE8

If this exit is enabled, it is taken whenever a program interrupt occurs due to a data exception within SORT. The data exception may be induced by an incorrect digit or sign in a packed decimal, signed decimal, or zoned decimal collating field. Normally the SORT program terminates in such circumstances; however, if this exit is enabled, the user's subroutine may attempt to rectify the error and resume processing.

The parameter list contains the addresses of two regions containing the records that were being compared when the exception occurred. The first halfword of each region contains a value of 4 plus the length of the text in that region. The text begins at an address 4 bytes higher than that of the region. The regions are aligned on a halfword boundary.

Either or both of the records may contain incorrect data. It is the user's responsibility to discover and correct the erroneous information. Note that corrections must be made to the erroneous data in the regions passed to the subroutine. Note too that the lengths of the records may not be altered.

The subroutine's return code is significant. If the code 0 is returned, processing will resume and the records' comparison will be attempted again. If a code greater than 3 is returned, processing terminates and a return code of 4 is passed to the program that called SORT or a signoff is produced if the control statement contains the SIG parameter.

Exit SORTE9

If this exit is enabled, it is taken just prior to SORT's return to its calling program or the operating system.

There are two parameters passed to the user's subroutine. The first word of the parameter list points to a two-word array containing the number of records processed and the number of output records. The second word of the parameter list points to a halfword containing the return code that the SORT program would pass to its calling program.

The subroutine's return code is passed to SORT's calling program or if the return code is greater than zero and the control statement contains the SIG parameter, a signoff is produced.

## OPTIONAL ENTRANCE

The SORT program can be entered at one point other than its normal entry point. This entrance, SORTEA, permits the user to dynamically enable or disable the exits SORTEO through SORTE9.

The optional entrance may be called at any time via the OS type (I) $S$ linkage.

The SORT program will return control to the calling program via the OS (I) S return.

## Entrance SORTEA

The entrance expects its parameter list to contain two pointers. The first word of the list should point to a word containing the integer number of the exit to be enabled or disabled, e.g., X'00000009' for SORTE9. The second word of the list should point to a word containing either 0 or the address of the subroutine that is to be called at the time of the exit: 0 indicates that the exit should be disabled. Both the exit's number and address should be aligned on fullword boundaries.

The return code from SORTEA is meaningful. A return code of 0 indicates that the specified exit has been enabled or disabled. A return code of 4 indicates that a parameter is invalid.

The following FORTRAN subroutine, SORTE2, deletes all records preceding a record containing the characters "9999" in columns 1 through 4. It then calls SORTEA to disable itself.

```
    SUBROUTINE SORTE2(REGION,LENGTH,*,*)
    INTEGER*2 REGION(40),NN/'99'/,LENGTH
    IF (REGION(1).NE.NN) GO TO 2
    IF (REGION(2).NE.NN) GO TO 2
    CALL SORTEA(2,0,&8)
    RETURN
    LENGTH = 0
    RETURN
RETURN 2
END
```


## SORT TIMING ESTIMATES

The following graphs depict several statistics of sampled sorts on the Amdahl $470 \mathrm{~V} / 8$ computer. Each graph pertains to a specific logical record length and delineates the CPU time and the CPU storage virtual memory integral (VMI) as a function of the number of logical records sorted. Specifically, the abscissa denotes the number of logical records and the ordinate the CPU time in seconds and the CPU storage VMI in page-seconds. The upper, middle, and lower solid lines represent the CPU time for unblocked records and block sizes equal to the record length times four and 32,000 bytes, respectively; while the upper, middle, and lower dashed lines represent the CPU storage VMI for unblocked records and block sizes equal to the record length times four and 32,000 bytes, respectively.

Each of the sampled sorts ordered a data set on 20 contiguous bytes in each record, specifically "B,A,1,20". The data within the collating fields were generated via a method that did not bias the timing estimates.

Each of the sampled sorts read a single data set from a tape and produced a single data set on a tape. Both data sets were recorded at 6250 bpi. The formats of the data sets were type $F$ for unblocked records and type FB for blocked records. The record length was 20, 80, 320, or 1280 bytes, and records were unblocked or blocked with a block length equal to the record length times four or 32,000 bytes. Blocking and deblocking occurred in the sort program and not in the magnetic tape device support routines. The control statement specified the exact record and block lengths and the exact number of records to be sorted.

Though the majority of sorting statistics should fall within the depicted range, deviations may occur. Any of the following factors can adversely affect the sort's performance. These factors, as listed, are ordered by increasing magnitude of the degradation to be expected.
bit, fixed-point, floating-point, or zoned decimal collation sequence collation with more than one intermediate file, ARL
reverse collation
defined sequence collation
signed decimal collation
alignment collation
multiple fields
using exit SORTE1 or SORTE4
using exit SORTE2 or SORTE3
using exit SORTE5
(It should be noted, however, that the performance produced by multiple fields involving the more efficient collating types or by some efficient exit subroutines may actually be better than the performance of the less efficient collating types.)

MTS 5: System Services


44 The SORT Utility Program



```
May 1983
```



SORT OPTIMIZATION

Among the variables that may affect the performance of a sort are the number and type of collating fields defined, the original distribution of the source data, the blocking factors of the source and sink data sets, the number, type, and efficiency of the exits employed, and the load upon the system.

Number and Types of Collating Fields

The mean time expended in the comparison of two records is dependent on the number and types of collating fields defined. Consequently it is advantageous to minimize the number of collating fields and to use the more efficient types of collating fields whenever possible.

Following is a list of collating field types ordered by decreasing efficiency. Those fields appearing on the same line have similar characteristics.
binary, character, packed decimal, sequence, or length
bit, fixed-point, floating-point, or zoned decimal
sequence with more than one intermediate file
defined sequence
signed decimal
alignment
call

Following is a list of equivalent collating field definitions. In each case the second definition (that following ::=) is superior.

```
C,A,3,1,C,A,4,5,C,A,9,2 ::= C,A,3,8
S,A,4,5 (and data are all >= 0) ::= B,A,4,5
Z,A,3,5 (and data are all >= 0) ::= B,A,3,5
P,A,100,8 (and data are all >= 0) ::= B,A,100,8
F,D,7,10 (and data are all >= 0) ::= B,D,7,10
FL,A,13,9 ::= FL,A,13,8
```


## Distribution of the Source Data

The algorithms used in the sort are slightly sensitive to the distribution of the source data. A slight degradation in performance may be experienced during the reverse collation of large data sets.

Blocking Factors of Data Sets

Perhaps the easiest and most effective method to increase the performance of a sort is to increase the blocking factors of the source and sink data sets. For voluminous sorts, sequential files with block lengths approaching 32,767 bytes are effective.

Depending on the record length, statistics indicate that the mean, total CPU time expended while processing data sets with only four records per block may be as small as one-third the time expended while processing the equivalent unblocked data sets. Reductions of CPU time to as little as one-tenth of the time required for unblocked data sets may be accomplished by maximum blocking. Commensurate savings occur in both the CPU storage VMI and the elapsed time.

Number, Type, and Efficiency of Exits

The degradation induced by enabled exits is usually proportional to the number of times the user's subroutines are called and to the CPU time expended by the subroutines.

The degradation induced by exits SORTE0 and SORTE6 through SORTE9 is negligible since each exit is taken but a few times during the course of the data's processing.

The degradation induced by exits SORTE1 through SORTE4 can be significant. SORTE1 and SORTE4 are taken for each block of data that is read or written while SORTE2 and SORTE3 are taken for each record that is read or written.

The degradation induced by exit SORTE5 is significant. SORTE5 can be called at least once for every comparison between two records. If SORTE5 were called once for every comparison, it would be called approximately $n(\log (b a s e ~ 2) ~ n) ~ t i m e s, ~ n ~ b e i n g ~ t h e ~ n u m b e r ~ o f ~ r e c o r d s ~$ processed.

The performance of all subroutines can be improved by assuring that they do not call the subroutines GETSPACE or FREESPAC whenever they are entered. Note that the ENTER and EXIT macros may generate calls to GETSPACE and FREESPAC.

## System Load

The time elapsed between the initiation and termination of a sort is dependent on the load supported by the system. Since terminal and tape users are charged for real time, they may wish to avoid initiating large sorts during the day on weekdays.

## PROCESSING MAGNETIC TAPES

The processing of magnetic tapes warrants discussion for two reasons: first, because of the confusion created by the existence of blocking facilities in both the SORT program and the system magnetic tape device support routines; and second, because the majority of large data sets reside on magnetic tape.

Both the SORT program and the MTS tape routines possess blocking and deblocking facilities. The facilities are alike save for SORT's inability to process records longer than 32,759 bytes and type FB blocks longer than 32,763 bytes.

Blocking or deblocking should occur but once while processing any record, hence the blocking facilities of either the MTS tape routines or the SORT program should be disabled. It is generally recommended that blocking be disabled in the MTS tape routines. In this case, the SORT program can perform the blocking (or deblocking) at a considerable saving in CPU time because a call to the MTS I/O routines is required only once per block instead of once per record.

Blocking can be disabled in the MTS tape routines by specifying BLOCKING=OFF via either the MOUNT command or the CONTROL command. ${ }^{1}$ If a tape is labeled, the FORMAT, LRECL, and SIZE parameters given to the MOUNT command should parallel those describing the data set in the SORT program.

Blocking may be disabled in the $S O R T$ program by referring to records as structure $U$ with appropriate record and block lengths: the record and block lengths pertain to the length of the deblocked record.

If, after a sort is completed, it is discovered that the output was damaged because the MTS tape routines truncated, lost, or reblocked the output, it may be possible to restart the sort using the RES parameter, which is described in the section entitled "Additional Parameters."

The SORT program neither positions or rewinds a tape before processing ${ }^{2}$ nor does it rewind a tape or write an end-of-file after processing. Consequently, it is the responsibility of the user to explicitly perform these functions whenever they become necessary.
${ }^{1}$ See the descriptions of the MOUNT and CONTROL commands in MTS Volume 1, The Michigan Terminal System.
${ }^{2}$ See, however, the description of the TPS parameter in the section entitled "Additional Parameters."

## STATISTICS AND DIAGNOSTICS

## Statistics

The SORT program prints the following statistics after processing the data unless the control statement contains the DEC parameter:
i. the number of logical records (not blocks) read and not deleted in the exit SORTE2 - The number of records read and written will be equal unless records are removed as a result of the DEL parameter or in the exits SORTE3 or SORTE4.
ii. the number of intermediate data sets used to process the data - Certain sorts require the support of files for temporary data storage. The existence of these files is managed dynamically by the SORT program and the user need not concern himself with them. The presence of an asterisk indicates that files were created but not used.
iii. the number of logical records (not blocks) written and not deleted as a result of the DEL parameter or in the exit SORTE3, if this number is different from the number of records read - The number of records written includes records removed in the exit SORTE4.
iv. the average length of the input records, if the ARL parameter is specified - This is the average length required within the SORT program for each data record and may be greater than the actual average length of these records.

## Diagnostics

The SORT program will issue diagnostics whenever it encounters an error in a control statement, in a data set, in a collating field, or from a user's subroutine. Any of the following may be the source of an error:
i. an incorrect parameter in the control statement, e.g., HI= or $S=Q$ or $I=, ., 2$ or $I=X, U, 80, \underline{90}$ or $O=Y, U, 100,100, Z, \underline{F}, 100,100$
ii. a collating field which extends beyond the greatest record length specified in the control statement
iii. incorrect data in a packed decimal, zoned decimal, or signed decimal collating field
iv. a record whose length exceeds its declared length
v. an incorrectly structured type V, VB, VS, or VBS record
vi. an error detected while reading or writing a data set - The error could be induced by the data's storage medium, e.g., a faulty magnetic tape; by insufficient space allocated to a sink data set; or by the device processing the data set.
vii. the end of the output description is reached and another sink data set is needed - This may be caused by a nonzero return code from WRITE or a return code of 4 from the user's subroutine SORTE3.
viii. insufficient disk space to process the data - This error

```
    occurs only in the sort mode.
ix. an incorrect parameter list passed to the SORT subroutine
x. an out-of-sequence record in a merge data set
xi. an error return from a user's subroutine
xii. an attempt to continue after certain program interrupts or
    input/output errors which occur outside of the SORT program
    and for which an appropriate exit was enabled before entry
    to the SORT program
```

```
SORT EXAMPLES
```

Example 1: Assume that 1000 cards form a data set. Each card bears the name of a city and state left-justified in columns 1 - 50 and the city's population right-justified in columns 65-75.

Typical data records are
CORPUS CHRISTI, TEX.
167690
NEW YORK, N.Y.
7781984
An alphabetized list of cities is needed.
\$RUN *SORT GUSER=*SOURCE*
SORT=CH,A,1,50 INPUT=*SOURCE*, U, 80, 80
OUTPUT=*SINK*,U,80,80 REC=1000 END
data
\$ENDFILE
or \$RUN *SORT SCARDS=*SOURCE* SPUNCH=*SINK* PAR=•
data
\$ENDFILE
or \$RUN *SORT GUSER=*SOURCE* SPUNCH=*SINK*
\$ENDFILE
data
...
\$ENDFILE
Example 2: Assume that the same data reside in a sequential file named SIN.
\$RUN *SORT SCARDS=SIN SPUNCH=*SINK* PAR=E
Example 3: A list of cities is needed. The cities should be ordered by decreasing population, and where equal populations occur, the cities should be ordered alphabetically. Two tapes are mounted with the pseudodevice names *DATA* and *BYPOP*. *DATA*'s records contain addi-
tional information to be included in the list. Its records have the format of SIN's but are blocked ten to the block - a total of approximately 2000 records. *BYPOP* will contain the sorted list. Its records should be type VB, a maximum of 10,000 bytes per block.
\$MOUNT
C9991 ON 9TP *DATA* BLK=OFF SIZE=800 'SORT INPUT' C9992 ON 9TP *BYPOP* RING=IN BLK=OFF SIZE=10000 'SORT OUTPUT' \$ENDFILE
\$RUN *SORT GUSER=*SOURCE*
S=C,D,65,11,,,,50 I=SIN,F,, ,*DATA*,FB, 80
O=*BYPOP*,VB, 80,10000 R=3500 E
Example 4: Assume that a printed list of *BYPOP*'s contents is needed.

```
$MOUNT C9992 ON 9TP *BYPOP* BLK=OFF SIZE=10000 'SORT OUTPUT'
$RUN *SORT GUSER=*SOURCE* SCARDS=*BYPOP* SPUNCH=*SINK*
C I=,VB,80,10000 END
or \$MOUNT C9992 ON 9TP *BYPOP* BLK=OFF SIZE=10000 'SORT OUTPUT' \$RUN *SORT SCARDS=*BYPOP* SPUNCH=*SINK* PAR=C I=,VB, 80
```

Example 5: A tape, mounted with the pseudodevice name *BYPOP2*, contains data similar to that appearing in *BYPOP*. *BYPOP2*'s records are ordered on the same fields as *BYPOP*'s records and are blocked 100 records to the block.

A unified list of cities ordered by decreasing population is desired. If two cities have equal populations, then the following rules apply. a) If the two records containing the cities' data were read from different data sets, then the record read from *BYPOP* should precede that read from *BYPOP2*. b) If the two records containing the cities' data were read from the same data set, they should be alphabetized by name.

```
$MOUNT
C9992 ON 9TP *BYPOP* BLK=OFF SIZE=10000 'SORT OUTPUT'
C9993 ON 9TP *BYPOP2* BLK=OFF SIZE=8000 'SORT OUTPUT #2'
$ENDFILE
$RUN *SORT GUSER=*SOURCE*
O=*SINK* I=*BYPOP*,VB,80,10000,*BYPOP2*,
FB,80,8000 M=,D,65,11,
SE,A
$ENDFILE
```

Example 6: Suppose that a file named SIN2 contains data akin to that in SIN. The 5000 records in SIN2 contain city and state names leftjustified in bytes 1 - 50 and the cities' populations left-justified in columns 65-75.

Typical data records in SIN2 are

```
WATERBURY, CONN.
1 0 7 1 3 0
LOS ANGELES, CALIF.
2479015
```

A list of cities arranged by decreasing population is needed.
\$RUN *SORT GUSER=*SOURCE*
S=AL,D,65,11,CH,D,65,11 I=SIN2 O=*SINK* R=5000 E
Example 7: Assume that 2000 cards form a data set. The first card of each pair of cards is identical to those described in example 1. The second card of each pair contains the city's area in columns 10-15.

Typical data records are
CORPUS CHRISTI, TEX.
167690
37
NEW YORK, N.Y.
7781984
300
An alphabetized list of cities is needed; however, the pairing of the cards must be preserved.

```
$CRE -T1 SIZE=5000
$CRE -T2 SIZE=5000
$RUN *SORT GUSER=*SOURCE* SPUNCH=-T1@\negTRIM
C I=,F O=,FB,80,160
$ENDFILE
data
...
$ENDFILE
$RUN *SORT GUSER=*SOURCE*
I=-T1@ \TRIM,F,160
O=-T2@\negTRIM,F,160 R=1000 E
$RUN *SORT GUSER=*SOURCE* SPUNCH=*SINK*
C I=-T2@ \TRIM,FB,80,160
$ENDFILE
```

Example 8: An alternate method to achieve the preceding results employs the exits SORTE1 and SORTE4.

Suppose the FORTRAN subroutines illustrated in the sections describing the exits SORTE1 and SORTE4 are compiled, and that the object modules reside in files named ElOBJ and E4OBJ. Assume that these subroutines are to be tested on a restricted data set.

```
$RUN *SORT+E1OBJ+E4OBJ GUSER=*SOURCE* 5=*SOURCE* 7=*SINK*
I=*DUMIMY*,F,160
O=*DUMIMY*,F,160
R=1000 CHK E
data
    ...
$ENDFILE
```

Example 9: Suppose that the data derived from a survey of magazine subscribers are encoded in a set of 20 -byte records residing in a file named Q. Each of the 2000 records pertains to a single subscriber.

The subscriber's sex is encoded in the 4 th bit of the 6 th byte of each record (0 indicates a male and 1 a female), and his preference for Saturday Review is encoded in the 0th bit of the 7 th byte of each record (1 indicates that he subscribes to the magazine and 0 indicates that he does not subscribe). A male subscriber might be encoded as X'87AF'.

A list of subscribers is needed. Those subscribing to Saturday Review should precede the nonsubscribers and males should precede females.

```
$RUN *SORT GUSER=*SOURCE*
S,BT,D,7,128,BT,A,6,8 I=Q,,20
O=*SINK*,,20 R=2000 E
```

Example 10: Assume the data records described in example 1 contain additional data in columns 55-60: the net change in population during the last year. The direction of the change is indicated by a sign which precedes the right-justified magnitude of the change.

Typical data records are

| CORPUS CHRISTI, TEX. | +4976 | 167690 |
| :--- | ---: | ---: |
| NEW YORK, N.Y. | -12856 | 7781984 |

A list of cities arranged by increasing growth is desired.
\$RUN *SORT SPUNCH=*SINK* PAR=S=SD,A,55,6 R=1000
data
\$ENDFILE

Example 11: Assume that a set of records has identifying numbers left-justified in columns 1-10. The numbers consist of one or more digits followed by a suffix of zero or more alphabetic characters. It is desired to list the records such that 1 precedes 1A, which precedes 1AA, which precedes 1AB; 1Z precedes 2; 9ZZ precedes 10; etc.

```
$RUN *SORT GUSER=*SOURCE* SPUNCH=*SINK*@\negCC
S=DS (X), , 10, CH, , ,10
DS=XABCDEFGHIJKLMNOPQRSTUVWXYZ A01234567890X
$ENDFILE
data
$ENDFILE
```

Example 12: Suppose that it is desired to list 1000 records with columns 5-21 in the ascending sequence illustrated by the second examples in the sections describing the exits SORTE2 and SORTE3. A less efficient method than using the exits is to utilize a defined sequence.

Assume, also, that a listing on SERCOM of the structures, record lengths, block lengths, and names of the source and sink data sets is desired.

```
$RUN *SORT GUSER=*SOURCE* SPUNCH=*SINK*
S=D("),,5,17 D="•&.<(+|&!$*); --/,%_>?:#@'="0123456789" R=1000 L E
data
$ENDFILE
```

Example 13: A file named STAT contains unblocked records which are supposed to be alphabetized according to the data in columns 1-32. It is desired to verify that the records are indeed alphabetized.
\$RUN *SORT PAR=MERGE=CH,A,1,32 I=STAT DEL=DUP,SINGLE
If any records are out of order, an error message will be produced. (The use of DEL=DUP,SINGLE is to improve efficiency by discarding all output records.)

Example 14: Assume that it is also desired to list the first record of each group of records having the same data in columns 1 - 32 .
\$RUN *SORT PAR=M=,, 32 I=STAT O=*SINK*@NOCC DEL=T,S
Example 15: A listing of the ordering of a set of 121 characters according to the ascending, character collating sequence is needed. The list is to contain 41 characters per line. (Note that the ordering of the special characters depends on the hexadecimal representation of the particular character set used by the input or output device. ${ }^{1}$ )

```
$RUN *SORT PAR=S=C,,,1 I=*SOURCE*,FB,1 O=*SINK*@NOCC,FB,,41 R=121
- +•十
    |@#$%न&*()_+QWERTYUIOP!]
ASDFGHJKL:"}>ZXCVBNM?
    1234567890-=qwertyuiop$[
asdfghjkl;'{<zxcvbnm,./
$ENDFILE
```

yields
\$. $<(+\mid \&!\$ *) ;\urcorner-/, \%>?: \# @^{\prime}=" a b c d e f g h i\{\leq 1++$

] $\neq-A B C D E F G H I J K L M N O P Q R S T U V W X Y Z 0123456789$
${ }^{1}$ See the descriptions of the ASCEBC and EBCASC subroutines in MTS Volume 3, System Subroutine Descriptions, and the appropriate appendices in MTS Volume 1, The Michigan Terminal System, and MTS Volume 4, Terminals and Tapes.

THE SORT SUBROUTINE

## Introduction


#### Abstract

The SORT subroutine has four entry points, SORT, SORT2, SORT3, and SORT4. Each entry point was designed to fulfill a unique set of requisites: the SORT entry point duplicates the capabilities of the SORT program; the SORT2 entry point sorts data sets that reside in virtual memory (virtual data sets); the SORT3 entry point sorts virtual data sets in conjunction with a "tag" virtual data set; ${ }^{1}$ and the SORT4 entry point sorts virtual data sets of 4 -byte addresses according to attributes of the data referenced by the addresses.


Return Codes

The SORT subroutine's return code should be interrogated by the calling program. A return code of 0 indicates that processing was successful and a return code of 4 indicates that an error was detected.

## The Entry Point SORT

When entered at SORT, the subroutine's capabilities and restrictions parallel those of the SORT program. The only notable exceptions involve the manner in which the control statement is passed to the subroutine and the subroutine's ability to process data sets that reside in virtual memory (virtual data sets).

## Calling Convention

The subroutine may be called from any program written in a language utilizing the OS type (I) S linkage, e.g., FORTRAN; or may be called through an appropriate interface in other cases, e.g., PLCALL.

Prototype calling sequences follow:
Assembly - CALL SORT, (cstmt[,\{unit|vds|num\}]...)
FORTRAN - CALL SORT(cstmt[,\{unit|vds|num\}]...[,\&err])

```
1^A description of a tag data set follows the section titled "The Entry
    Point SORT3."
```

$$
\begin{aligned}
& \mathrm{PL} / \mathrm{I}-\quad \mathrm{CALL} \operatorname{PLCALL}(S O R T, n, \text { cstmt } \\
& {\left[, \operatorname{ADDR}\left(\left\{\text { unit|vds|num\})]...); }{ }^{1}\right.\right.\right.}
\end{aligned}
$$

The parameter list passed to the subroutine contains a pointer to the control statement; an optional list of file/device usage blocks, logical unit numbers, and/or virtual data sets; and an optional error exit.

## The following substitutions are necessary:

Replace "cstmt" by the location of the control statement. The entire control statement must be passed as a parameter since it is never read via GUSER.

The control statement must be terminated by the END parameter followed by a blank or space, e.g., "END.".

Replace "unit" by the location of a pointer to a file/device usage block (FDUB) or the location of a fullword integer logical unit number. This parameter is an alternate means of identifying a data set to the SORT subroutine. If a data set's name is included in the control statement, its "unit" should be omitted from the parameter list.

If a "unit" is passed as a parameter, the data set's attributes must be defined in the control statement. If a data set is to be the source of data, its description should follow the INPUT keyword; otherwise, its description should follow the OUTPUT keyword.

The order in which the data sets are described in the control statement should correspond to the order in which the corresponding "units" appear in the parameter list. Thus the "unit" for the first data set to be described in the control statement should be the second entry of the parameter list, and the "unit" for the second data set should be the third entry in the parameter list.

The description of each data set for which a "unit" appears in the parameter list should conform to the specifications set forth earlier subject to the following amendment:

```
i. Replace "name" by *, e.g., I=*,VB,120,32767
```

Replace "vds" by the location of the 1st byte of the 1st record of the virtual data set to be processed by the subroutine. Note that each virtual data set must contain one contiguous block of type $\mathrm{FB},^{2} \mathrm{VB}$, or VBS records. If no virtual data set(s) exist, the parameter(s) should be omitted.
${ }^{1}$ See the description of the subroutine PLCALL in MTS Volume 7, PL/I in MTS.
${ }^{2}$ Because a considerable gain in efficiency may be realized, the entry point SORT2 should be used in preference to the entry point SORT if there are no other determining factors.

Each virtual data set to be processed must also be described in the control statement. The order in which the data sets are described in the control statement should correspond to the order in which their addresses appear in the parameter list. The description of each virtual data set should conform to the specifications set forth earlier subject to the following amendments:

```
i. Replace "name" by **, e.g., I=**,FB,4,400.
```

ii. Replace "structure" by FB, VB, or VBS. Records residing in a virtual data set must form a contiguous block of fixed or variable length records. Appendix B contains descriptions of these record structures. Note that no virtual data set's length may exceed 32,767 bytes.
iii. If the record length is not specified, a length of 80 bytes is assumed.
iv. The block length defaults to the record length of type $F B$ records and to the record length plus 8 of type VB and VBS records.

Replace "num" by the location of a positive, nonzero, fullword integer. This parameter is an alternate way of specifying a numeric value in the control statement to the SORT subroutine. If all numeric values are specified directly in the control statement or by default, the "num" parameter(s) should be omitted.

If a numeric value is passed as a parameter, its meaning must be indicated in the control statement by an asterisk. The value of the parameter is in effect substituted for the asterisk. The order in which the parameters are passed should correspond to the order in which the values would have appeared in the control statement. The ordering of the aggregate set of "unit," virtual data set, and numeric value parameters should correspond to the order in which they are referenced in the control statement.

The control statement should conform to the specifications set forth earlier subject to the following amendment:

```
i. Replace the numeric value in the control statement by *, e.g.,
    \(\mathrm{S}=\mathrm{CH}, \mathrm{A}, *\), 4
```

Replace "\&err" by the location (statement number or label) to receive control if an error is detected by the SORT subroutine. If no error return exists, then this parameter should be omitted.

Replace "n" by the number of arguments to be passed to the SORT subroutine. The number must possess the attributes FIXED BINARY(31).

## Restrictions

i. The entire control statement must be passed as a parameter.
ii. The control statement must be terminated by the END parameter followed by a blank or space, e.g., "END.".

Timing Estimates
The subroutine's CPU time and CPU storage VMI statistics for the entry point SORT are equivalent to those of the SORT program if the former are normalized by the time and storage used by the program calling the subroutine.

Examples
Example 1: A file named X contains 1900 80-byte records. The records will be alphabetized by the characters appearing in columns 10 through 29. File Y will receive the sorted data. A FORTRAN program follows:

CALL SORT('S=, 10,20 I=X O=Y R=2000 E ', \&88)
88 STOP
. . .
Example 2: This FORTRAN example is similar to the preceding example except that the control statement is passed as a parameter to the subroutine.

SUBROUTINE BAH (CSTMT)
INTEGER*2 CSTMT (40)
CALL SORT (CSTMT (2) , \&88)
. . .
88 CALL SYSTEM
END
Assuming that BAH's object module resides in a file named BAH.O, the RUN command would be
\$RUN BAH.O PAR=S=, 10,20 I=X O=Y R=2000 E.
Example 3: This FORTRAN example is the same as example 1 except that the records will be read from logical unit 1 and written to logical unit 3.

CALL SORT('S=, 10, 20 I=* $O=* R=2000 \mathrm{E}, 1,3, \& 88)$
88 STOP
STOP
...
Example 4: Assume that the following PL/I program creates a data set in a temporary file named -A. The records of this data set are to be ordered by an ascending character field in columns 12 through 17 and an ascending character field in columns 2 through 3. A temporary file named - $B$ will receive the sorted data.

```
DCL
            SORT ENTRY,
            PL1RC RETURNS (FIXED BIN(31)),
            F1 FIXED BIN(31) INIT(1) STATIC,
            1 CSTMT STATIC,
            2 A CHAR(24) INIT('S=,,12,6,,,2,2 I=-A,,46 '),
            2 B CHAR(11) INIT('O=-B,,46 R='),
            2 C PIC '(6)9',
            2 D CHAR(5) INIT(' END ');
            ...
/*SET NUMBER OF RECORDS IN THE CONTROL STATEMENT*/
            CSTMT.C=X;
/*CALL SORT*/
            CALL PLCALL(SORT,F1,CSTMT);
            IF PL1RC\neg=0 THEN DO;
                PUT FILE(SERCOM) LIST('**ERROR RETURN FROM SORT');
                RETURN;
            END;
```

            ...
    Example 5: In the following FORTRAN program the 4 -byte integers residing in a $50 x 100$ matrix, TAM, will be ordered into a descending sequence.

CALL SORT('S=FI,D, 4 I=**,FB, 4, 20000 O=**, FB, 4, 20000
C $\mathrm{R}=5000 \mathrm{E}$ ',TAM,TAM, \&88)
88 STOP
. . .

Note that the elements of a FORTRAN matrix may be accessed in their sorted order only if they are addressed by iterating on their row index before their column index.

Example 6: The 4-byte integers contained in a 100 x 100 matrix, X , and a 100 record file $Y$ (record structure $=F B$, record length $=4$, block length $=40$ ) will be sorted into a file named $Z$ (record structure $=F$ ).

CALL SORT('S=FI,,,4 I=**,FB, 4, 20000,**, FB, 4, 20000, Y, FB, 4, 40
C $\mathrm{O}=\mathrm{Z}, \mathrm{F}, 4,4 \mathrm{R}=11000 \mathrm{E}, \mathrm{X}(1,1), \mathrm{X}(1,51), \& 88)$
STOP
...

Note that the 32,767 -byte length restriction on virtual data sets forces $X$ to be referenced as two data sets.

Example 7: The NRELT, 4-byte integers in the array TBA are to be sorted and then written to logical unit 7. CALL SORT('S=F,,,4 I=**,FB,4,* O=*,F R=* E ',TBA, NRELT*4, C 7,NRELT,\&73) ... STOP . . .

The Entry Point SORT2

When entered at SORT2, the SORT subroutine is able to order an array of data elements, i.e., to sort a virtual data set onto itself. The subroutine is unable to read or write data sets resident on direct access devices, tapes, or other external media.

## Calling Conventions

The subroutine may be called from any program written in a language utilizing the OS type (I) S linkage, e.g., FORTRAN; or may be called through an appropriate interface in other cases, e.g., PLCALL.

Prototype calling sequences follow:

```
Assembly - CALL SORT2,(cstmt,loc1,loc2,len[,num]...)
FORTRAN - CALL SORT2(cstmt,loc1,loc2,len[,num]...[,&err])
PL/I - CALL PLCALL(SORT2,n,cstmt,ADDR(loc1),ADDR(loc2),
    ADDR (len) [, ADDR (num)] . . .); }\mp@subsup{}{}{1
```

The parameter list passed to the subroutine contains a pointer to the control statement, a description of an array to be sorted, and an optional error exit.

The following substitutions are necessary:
Replace "cstmt" by the location of the control statement. Only a description of the collating sequence, relevant additional parameters, and the END parameter should be included in the control statement.

The control statement must be terminated by the END parameter followed by a blank or space, e.g., "S=FI,D,1,4,CH,A,5,2 END.".
${ }^{1}$ See the description of the subroutine PLCALL in MTS Volume 7, PL/I in MTS.

Replace "loc1" by the location of the first element of the data set or array to be sorted.

Replace "loc2" by the location of the last element of the data set or array to be sorted.

Replace "len" by the location of the fullword integer length of each element in the data set to be sorted. The value of the length may range between 1 and 256 bytes.

Replace "num" by the location of a positive, nonzero, fullword integer. This parameter is an alternate way of specifying a numeric value in the control statement to the SORT subroutine. If all numeric values are specified directly in the control statement or by default, the "num" parameter(s) should be omitted.

If a numeric value is passed as a parameter, its meaning must be indicated in the control statement by an asterisk. The value of the parameter is in effect substituted for the asterisk. The order in which the parameters are passed should correspond to the order in which the values would have appeared in the control statement. The control statement should conform to the specifications set forth earlier and the restrictions below subject to the following amendment:

```
i. Replace the numeric value in the control statement by *, e.g.,
    S=CH,A, *,4
```

Replace "\&err" by the location (statement number or label) to receive control if an error is detected by the subroutine. If no error return exists, then this parameter should be omitted.

Replace "n" by the number of arguments to be passed to the SORT subroutine. The number must possess the attributes FIXED BINARY(31).

## Restrictions

i. Only a description of the collating sequence, relevant additional parameters, and the END parameter may be included in the control statement. Descriptions of the source and sink data sets and an estimate of the number of records to be sorted should not be included, i.e., the REC, MNR, INPUT, and OUTPUT keywords should not appear in the control statement. The mode must be SORT. The data types LE and $S E$ and the parameters CHK, DEL, LIO, RES, SIG, TPS, ARL, DIO, MBY, and PGS cannot be used.
ii. Only the exits SORTE5 and SORTE8 may be enabled. Note that the exits' parameter lists differ from the normal: the regions addressed contain only the records' text, i.e., they do not contain a 4-byte header.
iii. The entry point SORTEA may not be called.

Timing Estimates
Sample character sorts of up to one segment (1,048,576 bytes) of uniformly distributed, random data on the Amdahl 470V/8 indicate that the CPU times required by the subroutine for the entry point SORT2 range from approximately 12 microseconds to approximately 63 microseconds per array element depending on the number of elements and their length.

Examples
Example 1: In the following FORTRAN program the 4-byte integers residing in a $2 \times 100$ array, $X$, will be ordered into a descending sequence.

INTEGER*4 X $(2,100), F 4 / 4 /$
CALL SORT2 ('S=FI,D, 4 END ', X(1,1), X $(2,100), F 4, \& 88)$
. . .
88
STOP
. . .
Note that the elements of a FORTRAN array may be accessed in their sorted order only if they are addressed by iterating on their row index before their column index.

Example 2: The following PL/I program is analogous to that in example 1.

```
DCL
            SORT2 ENTRY,
            PL1RC RETURNS (FIXED BIN(31)),
            F4 FIXED BIN(31) INIT(4) STATIC,
            X(2,100) FIXED BIN(31) STATIC,
            CSTMT STATIC CHAR(14) INIT('S=FI,D,,4 END ');
            ...
/*CALL SORT2*/
            CALL PLCALL(SORT2,F4,CSTMT,ADDR(X (1,1)),ADDR(X (2,100)),
            ADDR(F4));
            IF PL1RC\neg=0 THEN DO;
                PUT FILE(SERCOM) LIST('**ERROR RETURN FROM SORT2');
                    RETURN;
            END;
```

            ...
    The Entry Point SORT3

When entered at SORT3, the SORT subroutine's capabilities are similar to those it possesses when entered at SORT2; however, it has the additional capability of permuting a "tag" virtual data set. The order induced on the elements of the tag data set is analogous to that induced on the elements of the source/sink data set: if the 10 th element of the source data set became the 1 st element of the sorted data set, then the 10th element of the tag data set would appear as its 1st element after the sort.

## Calling Conventions

The subroutine may be called from any program written in a language utilizing the OS type (I) S linkage, e.g., FORTRAN; or may be called through an appropriate interface in other cases, e.g., PLCALL.

Prototype calling sequences follow:
Assembly - CALL SORT3,(cstmt,loc1,loc2,len1,loc3,len2[,num]...)
FORTRAN - CALL SORT3 (cstmt,loc1,loc2,len1,loc3,len2[,num]...
[, \&err])
PL/I - CALL PLCALL (SORT3,n,cstmt,ADDR (loc1), ADDR (loc2), ADDR (len1), ADDR (loc3), ADDR (len2)
[, ADDR (num)]...); ${ }^{1}$
The parameter list passed to the subroutine contains a pointer to the control statement, a description of an array to be sorted, a description of a tag array, and an optional error exit.

The following substitutions are necessary:

Replace "cstmt" by the location of the control statement. Only a description of the collating sequence, relevant additional parameters, and the END parameter should be included in the control statement.

The control statement must be terminated by the END parameter followed by a blank or space, e.g., "S=FL,,,8 END.".

Replace "loc1" by the location of the first element of the data set or array to be sorted.

Replace "loc2" by the location of the last element of the data set or array to be sorted.
${ }^{1}$ See the description of the subroutine PLCALL in MTS Volume 7, PL/I in MTS.

Replace "len1" by the location of the fullword integer length of each element in the data set to be sorted. The value of the length may range between 1 and 256 bytes.

Replace "loc3" by the location of the first element in the tag data set or array.

Replace "len2" by the location of the fullword integer length of each element of the tag data set. The value of the length may range between 1 and 256 bytes.

Replace "num" by the location of a positive, nonzero, fullword integer. This parameter is an alternate way of specifying a numeric value in the control statement to the SORT subroutine. If all numeric values are specified directly in the control statement or by default, the "num" parameter(s) should be omitted.

If a numeric value is passed as a parameter, its meaning must be indicated in the control statement by an asterisk. The value of the parameter is in effect substituted for the asterisk. The order in which the parameters are passed should correspond to the order in which the values would have appeared in the control statement. The control statement should conform to the specifications set forth earlier and the restrictions below subject to the following amendment:

```
i. Replace the numeric value in the control statement by *, e.g.,
    S=CH,A, *,4
```

Replace "\&err" by the location (statement number or label) to receive control if an error is detected by the subroutine. If no error return exists, then this parameter should be omitted.

Replace "n" by the number of arguments to be passed to the SORT subroutine. The number must possess the attributes FIXED BINARY(31).

## Restrictions

i. Only a description of the collating sequence, relevant additional parameters, and the END parameter may be included in the control statement. Descriptions of the source and sink data sets and an estimate of the number of records to be sorted should not be included, i.e., the REC, MNR, INPUT, and OUTPUT keywords should not appear in the control statement. The mode must be SORT. The data types LE and $S E$ and the parameters CHK, DEL, LIO, RES, SIG, TPS, ARL, DIO, MBY, and PGS cannot be used.
ii. Only the exits SORTE5 and SORTE8 may be enabled. Note that the exits' parameter lists differ from the normal: the regions addressed contain only the records' text, i.e., they do not contain a 4 -byte header.
iii. The entry point SORTEA may not be called.

Timing Estimates

Sample character sorts of up to one segment (1,048,576 bytes) of uniformly distributed, random data on the Amdahl 470V/8 indicate that the CPU times required by the subroutine for the entry point SORT3 range from approximately 18 microseconds to approximately 119 microseconds per array element depending on the number of elements and their lengths.

Example
Example 1: In the following FORTRAN program the 8 -byte REAL numbers residing in a $1 \times 500$ array, $X$, will be ordered into an ascending sequence. The tag array, Y, consists of 5004 -byte integers.

```
                REAL*8 X(500)
```

                INTEGER*4 Y(500),F4/4/,F8/8/
                CALL SORT3('S=FL, , 8 END ', X(1), X(500),F8,Y(1),F4, \&88)
                ...
                88 STOP
                        ...
    
## The Entry Point SORT4

When entered at SORT4, the SORT subroutine is able to order an array of 4 -byte addresses, or pointers, according to attributes of the data elements to which the addresses point. The data elements themselves are not reordered or otherwise changed. The order induced on the elements of the address array is such that if the data elements were substituted for their respective addresses, they would form the collating sequence defined by the control statement. The subroutine is unable to read or write data sets resident on direct access devices, tapes, or other external media.

## Calling Conventions

The subroutine may be called from any program written in a language utilizing the OS type (I) S linkage, e.g., FORTRAN; or may be called through an appropriate interface in other cases, e.g., PLCALL.

Prototype calling sequences follow:
Assembly - CALL SORT4,(cstmt,loc1,loc2[,num]...)
FORTRAN - CALL SORT4 (cstmt,loc1,loc2[,num]...[, \&err])

```
PL/I - CALL PLCALL(SORT4,n,cstmt,ADDR(loc1),ADDR(loc2)
    [,ADDR (num)]...);'1
```

The parameter list passed to the subroutine contains a pointer to the control statement, a description of an array of 4 -byte addresses to be sorted, and an optional error exit.

## The following substitutions are necessary:

Replace "cstmt" by the location of the control statement. Only a description of the collating sequence, relevant additional parameters, and the END parameter should be included in the control statement. The description of the collating sequence refers to the data addressed by the elements of the array to be sorted.

The control statement must be terminated by the END parameter followed by a blank or space, e.g., "=,, 16 E.".

Replace "loc1" by the location of the first element of the fullwordaligned array of 4 -byte addresses to be sorted. The data elements referenced by the addresses need not be contiguous to each other, and more than one element of the address array may reference the same data element.

Replace "loc2" by the location of the last element of the fullwordaligned array of 4 -byte addresses to be sorted.

Replace "num" by the location of a positive, nonzero, fullword integer. This parameter is an alternate way of specifying a numeric value in the control statement to the SORT subroutine. If all numeric values are specified directly in the control statement or by default, the "num" parameter(s) should be omitted.

If a numeric value is passed as a parameter, its meaning must be indicated in the control statement by an asterisk. The value of the parameter is in effect substituted for the asterisk. The order in which the parameters are passed should correspond to the order in which the values would have appeared in the control statement. The control statement should conform to the specifications set forth earlier and the restrictions below subject to the following amendment:

```
i. Replace the numeric value in the control statement by *, e.g.,
    S=CH,A,* , 4
```

Replace "\&err" by the location (statement number or label) to receive control if an error is detected by the subroutine. If no error return exists, then this parameter should be omitted.

[^1] MTS.

Replace "n" by the number of arguments to be passed to the SORT subroutine. The number must possess the attributes FIXED BINARY(31).

Restrictions
i. Only a description of the collating sequence, relevant additional parameters, and the END parameter may be included in the control statement. Descriptions of the source and sink data sets and an estimate of the number of records to be sorted should not be included, i.e., the REC, MNR, INPUT, and OUTPUT keywords should not appear in the control statement. The mode must be SORT. The data types $L E$ and $S E$ and the parameters CHK, DEL, LIO, RES, SIG, TPS, ARL, DIO, MBY, and PGS cannot be used.
ii. Only the exits SORTE5 and SORTE8 may be enabled. Note that the exits' parameter lists differ from the normal: the regions addressed contain the data, not the addresses being sorted, and they contain only the text, i.e., they do not contain a 4-byte header.
iii. The entry point SORTEA may not be called.

## Timing Estimates

Sample character sorts of up to one segment of addresses (262,144), which reference uniformly distributed, random data, on the Amdahl 470V/8 indicate that the CPU times required by the subroutine for the entry point SORT4 range from approximately 14 microseconds to approximately 45 microseconds per array element depending on the number of elements.

Examples
Example 1: The array LISTPTR contains up to 100 addresses which point to the noncontiguous elements of a list. Each of the $N$ elements of the list contains a positive, fixed-point number in positions 21-24. LISTPTR is to be sorted so that when it is used to indirectly access the list elements, these numbers will be in ascending order.

|  | L | 1, N |  |
| :---: | :---: | :---: | :---: |
|  | BCTR | 1,0 |  |
|  | SLA | 1,2 | ( $\mathrm{N}-1$ )*4=DISPLACEMENT OF LAST ELEMENT |
|  | LA | 1, LISTPTR (1) | ADDRESS OF LAST ELEMENT |
|  | ST | 1,SRT4PAR+8 | SAVE IT IN PARAMETER LIST |
|  | LA | 1, SRT4PAR | ADDRESS OF PARAMETER LIST |
|  | CALL | SORT4 | SORT LISTPTR |
|  | LTR | 15,15 | CHECK RETURN CODE |
|  | BNZ | ERRRTN | ERROR IF RETURN CODE IS NOT ZERO |
| ERRRTN | $\dot{\text { CALL }}$ | ERROR |  |
| LISTPTR | DS | 100A |  |
| N | DS | F |  |
| SRT4PAR | DC | A (SRT4CS,LIS | TR, 0) |
| SRT4CS | DC | $C^{\prime} \mathrm{S}=\mathrm{B}, \mathrm{A}, 21,4$ | E |

Example 2: In the following FORTRAN program the array NAMES contains $N$ columns of 20-character names. The address of each column is placed in the array INDEX; the addresses are sorted and then converted to column indexes; and an alphabetical listing of the names is written.

INTEGER ADROF, BASE, $\operatorname{INDEX}(256), \operatorname{NAMES}(5,256)$
DO 1010 I=1,N
$\operatorname{INDEX}(I)=\operatorname{ADROF}(\operatorname{NAMES}(1, I))$
BASE=INDEX (1)
CALL SORT4 ('SORT=CH, A, 1,20 END ', INDEX (1), INDEX (N) , \&9999)
DO 1020 I=1,N
C
C IN THE FOLLOWING CALCULATION, 20=THE NUMBER OF BYTES IN A C COLUMN=ADROF $(\operatorname{NAMES}(1,2))-\operatorname{ADROF}(\operatorname{NAMES}(1,1))$.
C
C IN GENERAL, IF THE DIMENSIONS OF A FORTRAN ARRAY ARE
C DECLARED TO BE (D[1],D[2],...,D[N]) AND THE SUBSCRIPTS ARE
C (S[1],S[2],...,S[N]), THEN THE VALUE OF S[I] CAN BE
C CALCULATED FROM AN ADDRESS, ADR, BY THE FORMULA
C
C
C WHERE BASE=THE ADDRESS OF THE ELEMENT WITH ALL SUBSCRIPTS
C EQUAL TO 1, D[0]=THE LENGTH OF EACH ELEMENT OF THE ARRAY,
C AND ALL VARIABLES AND CONSTANTS ARE INTEGER.
C
$1020 \operatorname{INDEX}(\mathrm{I})=(\operatorname{INDEX}(\mathrm{I})-B A S E) / 20+1$
$\operatorname{WRITE}(6,2000)((\operatorname{NAMES}(I, \operatorname{INDEX}(J)), \quad I=1,5), J=1, N)$
2000 FORMAT (1X,5A4)
999
STOP

May 1983

## BIBLIOGRAPHY

Beus, H.L. "The Use of Information in Sorting." JACM 17(1970):482-95. Black, N.A. "Optimum Merging from Mass Storage." CACM 13(1970):745-49.

Brawn, B.S., Gustavson, E.S., Mankin, E.S. Sorting Performance in a Paged Virtual Memory, Report RC2435. Yorktown Heights, New York: IBM T.J. Watson Research Center, 1969.

Ford, L.R., Johnson, S.M. "A Tournament Problem." Amer. Math. Monthly 66(1959):387.

Friend, E.H. "Sorting on Electronic Computer Systems." JACM 3(1956):134-68.

Gassner, B.J. "Sorting by Replacement Selection." CACM 10(1967):89-93.
Gotlieb, C.C. "Sorting on Computers." CACM 6(1963):194-201.
IBM. OS Sort/Merge Program, Form GC28-6543. White Plains, New York: IBM Data Processing Division, 1973.

IBM. OS/VS2 MVS Data Management Services Guide, Form GC26-3875. White Plains, New York: IBM Data Processing Division, 1976.

Knuth, D.E. "Length of Strings for a Merge Set." CACM 6(1963):685-88.
Knuth, D.E. "Three Letters on Merging." CACM 6(1963):585.
Malcolm, Jr., W.D. "String Distribution for the Polyphase Sort." CACM 6(1963):217-20.

Shell, D.L. "Optimizing the Polyphase Sort." CACM 14(1971):713-19, 15(1972):28.

Stafel, L.E. "Tree Structures for Optimal Searching." JACM 17(1970):508-17.

Woodrum, L.J. "Internal Sorting with Minimal Comparing." IBM Systems Journal 8(1969):189-203.

## APPENDIX A: COLLATING FIELDS

The SORT program can order records by any of several schemes. A sequence can be based on attributes of the data contained in the records, on an attribute of the records, on an attribute of the data set in which the records reside, or on some combination of these attributes.

Any of the following operands are accommodated by the SORT program and can be used to induce the desired order.

Alignment fields (AL)
Alignment fields are composed of 1 to 4095 bytes or characters. All bit permutations within a field are permitted.

In the alignment sequence, fields are ordered by the number of trailing blanks that occur in each field. If "x" represents any character and "Y" represents any character other than a blank, then the ascending alignment collating sequence is
-.... y...." xy..... ... x...xy..... ... x...xy. x....xy

See example 6 in the section entitled "Sort Examples."

Binary and character fields (BI \& CH)
The binary or character field is the most common of all collating fields.

Binary, or character, fields are composed of 1 to 256 -bit bytes or characters. All bit permutations within a field are permitted.

In the ascending binary, or character, collating sequence, fields are ordered by increasing binary value. The more common graphics are ordered

```
    -&.<(+|&!$*);ᄀ-/,%_>?:#@'="
        abcdefghijklmnopqrstuvwxyz
            ABCDEFGHIJKLMNOPQRSTUVWXYZ
                0123456789
```

See examples $1,2,3,7,11,13,14$, and 15 in the section entitled "Sort Examples."

Bit fields (BT)
Bit fields are composed of the 8 bits of a single byte. All bit permutations within a field are permitted.

The length specified in the field's description is used as a mask. The logical product, AND, of a record's bit field and the mask is used in the records' comparison. For a specific mask, the collating sequence is identical to the binary sequence.

See example 9 in the section entitled "Sort Examples."

CALL fields (CA)
Call fields are defined by the user and are used in conjunction with exit SORTE5. If the comparison of two records reaches the level at which CA is encoded, the user's subroutine, SORTE5, is entered. The subroutine's return code indicates the result of the records' comparison. If the return code implies equivalent collation, comparisons defined subsequent to CA are made as necessary.

See the description of the exit SORTE5 for further information about the CALL field.

Defined sequence fields (DS (i))
Defined sequence fields are similar to binary and character fields except that the user defines the collating order.

Defined sequence fields are composed of 1 to 2568 -bit bytes or characters. All bit permutations within a field are permitted.

The collating sequence for a defined sequence field is defined by the DS keyword parameter that uses the same character for a delimiter as that which appears in the parentheses after the collating field type "DS". Each defined sequence field may use a separate collating definition, or several fields may share the same definition. If "s" represents the collating sequence defined by the string in the $D S$ keyword parameter (with possibly some bit combinations defined to compare equal), $X^{\prime} 00$ ' is not in "s", and "r" represents the remaining bit combinations in their normal character, or binary, collating relationship to each other, then the full ascending collating sequence is

$$
x^{\prime} 00^{\prime} \mathrm{sr}
$$

If $X^{\prime} 00$ ' is in "s", then the ascending collating sequence is
sr
In this case, if the field extends beyond the end of a record, the resulting order may not be as expected because short records are padded with hexadecimal zeros ( $\mathrm{X}^{\prime} 00^{\prime}$ ).

```
See examples }11\mathrm{ and 12 in the section entitled "Sort Examples."
```

Fixed-point fields (FI)
Fixed-point fields are composed of a sign and an integer. They may range from 1 to 260 bytes in length and may be aligned on any byte boundary within a record.

Positive numbers are represented by a sign bit of zero followed by a binary integer.

Negative numbers are represented by a sign bit of one followed by a complemented binary integer. The complemented integer is formed by inverting each bit in the integer and adding 1 to the result.

The halfword, fixed-point representations of 2 and -3 are $X^{\prime} 0002$ ' and $X^{\prime} F F F D^{\prime}$.

The ascending fixed-point collating sequence is
$X^{\prime} 8000 . .00^{\prime} . . . X^{\prime} 7 F F F . . F^{\prime}$
See examples 5, 6, and 7 in the section entitled "The Entry Point SORT" and the examples in the section entitled "The Entry Point SORT2."

## Floating-point fields (FL)

Floating-point fields are composed of a sign, a characteristic, and a fraction. They may range from 2 to 16 bytes in length and may be aligned on any byte boundary within a record.

Positive numbers are represented by a sign bit of zero followed by a 7 -bit binary characteristic and a 1 to 15 byte hexadecimal fraction. The number is formed from the product of a power-of-16 and the fraction. The power to which 16 is raised is calculated by subtracting 64 from the characteristic. The fraction consists of a series of hexadecimal digits with an assumed radix point to the left of the high-order digit. If the fraction is 8 to 15 bytes in length, the eighth byte (ninth byte of the field) is ignored. Consequently, the fraction is treated as though its ninth byte immediately followed its seventh byte.

Negative numbers are identical to their equivalent positive numbers save for a sign bit of one.

The 4 -byte floating-point representations of 8 and -12 are $X^{\prime} 41800000$ and $X^{\prime} \mathrm{C} 1 \mathrm{C00000}$.

If "x" represents any hexadecimal digit, then the ascending floatingpoint collating sequence is
$X^{\prime}$ FFFFFFFFFFFFFFFFxxFFFFFFFFFFFFFF'... $X^{\prime}$ 7FFFFFFFFFFFFFFFXXFFFFFFFFFFFFFF'

May 1983

See example 1 in the section entitled "The Entry Point SORT3."

Length fields (LE)
Length fields are composed of a halfword record length generated by the SORT program.

In the ascending length sequence, records are arranged by increasing record length.

Packed decimal fields (PD)
Packed decimal fields are composed of 1 to 31 digits and a sign. Each digit occupies 4 bits of a byte, i.e., two digits per byte.

The low-order 4 bits of the last byte specify the sign of the field: $X^{\prime} D^{\prime}$ denotes a negative field while $X^{\prime} C^{\prime}$ denotes a positive field.
-20 and 9 may be represented as $X^{\prime} 020 D^{\prime}$ and $X^{\prime} 9 C^{\prime}, ~ r e s p e c t i v e l y$.
The ascending packed decimal collating sequence is

$$
\begin{aligned}
& \text { X'9999999999999999999999999999999D'... } \\
& \text { X'9999999999999999999999999999999C' }
\end{aligned}
$$

Sequence fields (SE)
The SE parameter may be used to maintain the existing order of records that collate equivalently. When sorting, ascending sequence "collating" orders the records in the same order in which they are read from the source data sets. When merging, ascending sequence "collating" orders equivalent records in the order in which their data sets are described in the control statement. When equivalent merge records are read from the same source data set, they are ordered, for both ascending and descending sequence "collating," in the same relationship as they are read.

See example 5 in the section entitled "Sort Examples."

Signed decimal fields (SD)
The signed decimal field is probably the second most prevalent field.
Signed decimal fields are composed of an optional sign and 1 to 16 digits or blanks. The digits must be right-justified in the field and the sign, which may be "+" or "-", may occupy any location of the field which is to the left of the digits. If no sign is present, a "+" is assumed.

The ascending signed decimal sequence is
-9999999999999999 . .. +9999999999999999

See example 10 in the section entitled "Sort Examples."

Zoned decimal fields (ZD)
Zoned decimal fields are composed of 1 to 16 digits and a sign. Each digit, except the last, occupies 1 byte and may be depicted as $X^{\prime} F^{\prime}$, "x" having a value from X'0' to X'9'.

The high-order 4 bits of the last digit specify the sign of the field: $X^{\prime} D^{\prime}$ denotes a negative field while $X^{\prime} C^{\prime}$ and $X^{\prime} F^{\prime}$ denote positive fields.
-20 and 9 may be represented as $X^{\prime} F 2 D 0^{\prime}$ and $X^{\prime} C 9^{\prime}, ~ r e s p e c t i v e l y$.
The ascending zoned decimal collating sequence is
X'F9F9F9F9F9F9F9F9F9F9F9F9F9F9F9D9'...
X'F9F9F9F9F9F9F9F9F9F9F9F9F9F9F9C9'

## APPENDIX B: RECORD STRUCTURES

The SORT program can process data sets containing records structured by any of eight methods. The following symbols usually represent the structures: U, F, V, VS, FB, VB, VBS, and FBS.

The name associated with each structure alludes to characteristics of the structure's record length and blocking scheme. U, F, and $V$ denote length attributes, while $B$ and $S$ specify blocking attributes.

Undefined length (U)
The $U$ structure is the most common of all record structures.
Type U records are of variable length and are not blocked. Each block of data transmitted by the device support routine associated with a data set is treated as a record.

The declared record and block lengths of type u records must be equal. For a source data set, the declared lengths designate the length of the longest record in the data set. For a sink data set, the declared lengths indicate the length of the longest record to be written. Truncation of output records occurs if the actual record length exceeds the declared length.

Fixed length (F)
The $F$ structure is probably the second most common record structure.
Type F records are of fixed length and are not blocked. They are similar to type $U$ records save for an assumption implicit in their length specifications.

The declared record and block lengths of type $F$ records must be equal. For a source data set, the declared lengths specify the length of each record existing in the data set. Extension (padding with blanks) of input records occurs if the actual record length is less than the declared length. For a sink data set, the declared lengths indicate the length of each record to be written. Truncation or extension of output records occurs if the actual record length is not equal to the declared length. Note that input records are never truncated.

Fixed length; blocked (FB)
Type $F B$ records are of fixed length and are blocked. Each block is composed of several concatenated type $F$ records.

In a source data set's declaration, the record length denotes the length of each record in the data set. If the actual length of the last
record in an input block is less than the declared length, extension occurs. The block length indicates the length of the longest block in the data set and must be an integral multiple of the record length.

In a sink data set's declaration, the record length specifies the length of each record to be written. If the actual length of an output record is not equal to the declared length, truncation or extension occurs. The declared block length must be an integral multiple of the record length.

## Fixed length; blocked; standard (FBS)

The specifications of type FBS and FB records are identical except for the rules pertaining to block formation. The actual length of each block of an FBS data set, except the last, must be equal to the declared block length. Both FB and FBS records are treated identically by SORT.

Variable length (V)
The $V$ structure occurs infrequently and is often mistaken as a substitute for the $U$ structure.

Type V records are of variable length and are not blocked. The text of each record is prefaced by a 4 -byte record descriptor (RD) and a 4 -byte block descriptor (BD). Thus each block looks like this

$$
\begin{array}{l|l|l}
\mathrm{BD} & \mathrm{RD} & \mathrm{TEXT}
\end{array}
$$

The first two bytes of the block descriptor contain the fixed-point, byte length of the block, including the block-descriptor length. The third and fourth bytes of the block descriptor contain zero. The first two bytes of the record descriptor contain the fixed-point sum of the byte lengths of the text and the record descriptor, i.e., the text length plus 4. The third and fourth bytes of the record descriptor contain zero.

The declared block length of type $V$ records must be 8 bytes greater than the declared record length. For source data sets the declared record length specifies the length of the longest record in the data set. For sink data sets the declared record length establishes the length of the longest record to be written. If the actual length of an output record exceeds the declared length, the record is truncated.

Note that the type V record length does not include the length of the record descriptor as it does in IBM's Operating System.

Variable length; blocked (VB)
Type vB records are of variable length and are blocked. Their structure is similar to that of type $V$ records.

The text of each record is prefaced by a 4 -byte record descriptor, and each block is prefaced by a 4-byte block descriptor. The block and record descriptors are the same as for type V records. Each block resembles the following:
BD | RD1 | TEXT1 | RD2 | TEXT2 | ...

In a source data set's declaration, the record length indicates the length of the longest record occurring in the data set, and the block length indicates the length of the longest block in the data set. The block length must be at least 8 bytes longer than the record length.

In a sink data set's declaration, the record length indicates the length of the longest record to be written into the data set, while the block length indicates the length of the longest block to be formed. The block length must be at least 8 bytes longer than the record length. If the actual record length of an output record exceeds its declared length, the record will be truncated.

Variable length; spanned (VS)
The VS structure, although relatively uncommon, does occur when unformatted output is written by a FORTRAN program.

Type VS records are of variable length and may occupy more than one block. Each block contains a record segment, which may be all or part of a record; and the segments are not blocked. The text of each segment is prefaced by a 4-byte segment descriptor (SD) and a 4-byte block descriptor. Each record resembles the following:

```
BD1 | SD1 | TEXT-SEG1
BD2 | SD2 | TEXT-SEG2
```

| BD3 | SD3 | TEXT-SEG3 |
| :--- | :--- | :--- |

The block descriptor is the same as for type V records. The first two bytes of the segment descriptor contain the fixed-point sum of the byte lengths of the segment text and the segment descriptor, i.e., the segment text length plus 4. The third and fourth bytes of the segment descriptor contain zero except for the seventh and eighth bits of the third byte. If the seventh bit, $X^{\prime} 02^{\prime}$, of the third byte is a one, a segment of the record precedes this segment. If the eighth bit, $X^{\prime} 01^{\prime}$, of the third byte is a one, a segment of the record follows this segment.

The declared block length of type VS records must be greater than 8 bytes and may be less than the declared record length. For source data sets the declared record length specifies the length of the longest record in the data set. For sink data sets the declared record length
establishes the length of the longest record to be written. If the actual length of an output record exceeds the declared length, the record is truncated.

Note that the type VS record length does not include the length of the segment descriptor as it does in IBM's Operating System.

Variable length; blocked; spanned (VBS)
Type VBS records are of variable length; they may occupy more than one block; and the segments are blocked. Their structure is similar to that of type VS records.

The text of each segment is prefaced by a 4-byte segment descriptor, and each block is prefaced by a 4-byte block descriptor. The block descriptor is the same as for type V records; the segment descriptor is the same as for type VS records. Thus blocks resemble this

| BD1 | SD1 | TEXT1 | SD2 | TEXT2-SEG1 |
| :--- | :--- | :--- | :--- | :--- |

BD2 | SD3 | TEXT2-SEG2
BD3 $\mid$ SD4 $\mid$ TEXT2-SEG3 $\mid$ SD5 $\mid$ TEXT3 $\mid$ SD6 $\mid$ TEXT4 $\mid$ SD7 $\mid$ TEXT5-SEG1

In a source data set's declaration, the record length indicates the length of the longest record occurring in the data set, and the block length indicates the length of the longest block in the data set. The block length must be greater than 8 bytes and may be less than the record length.

In a sink data set's declaration, the record length indicates the length of the longest record to be written into the data set, while the block length indicates the length of the longest block to be formed. The block length must be greater than 8 bytes and may be less than the record length. If the actual record length of an output record exceeds its declared length, the record will be truncated.

APPENDIX C: SUMMARY OF THE CONTROL STATEMENT

Prototype:
[COPY|[[SORT|쓰NERE] [=[[type], [aspect], [location], [length],]...
[type][,[aspect][,[location][,[length]]]]]]
[ [DS=delimiter[string]delimiter•]...DS=delimiter [string]delimiter]
[INPUT[=[[name], [structure], [record length], [block length],]...
[name] [, [structure] [, [record length] [, [block length]]]]]]
[OUTPUT[=[[name], [structure], [record length], [block length],]...
[name][,[structure][,[record length][,[block length]]]]]]
[additional parameter]...
[END]


Record structures:

| CODE | RECORD STRUCTURE |
| :--- | :--- |
| U | undefined length |
| F | fixed length |
| V | variable length |
| VS | variable length; spanned |
| FB | fixed length; blocked |
| VB | variable length; blocked |
| VBS | variable length; blocked; spanned |
| FBS | fixed length; blocked; standard |

Additional parameters:
CHK (exit check facility)
DEC (delete comments)
DEL=x[,x]... (delete output records)
LIO (list data set characteristics)
\{REC|MNR\} =x (number of records)
RES=x (restart)
SIG (sign off on error)
TPS[=\{x|name, name[,name]...\}] (tape-merge sort facility)

MTS 5: System Services

May 1983

82 The SORT Utility Program

THE DYNAMIC LOADER

ELEMENTARY LOADER TOPICS

## Introduction

Programs written in languages other than machine language must be translated into machine language and loaded into main memory before they can be executed. While a program is being debugged, it is useful to have language processors like *WATFIV and *SNOBOL4 that can translate a program, load it into main memory, and monitor its execution. However, once a program is debugged, it is useful to save it in translated form so that it need not be translated each time it is used.

If the translation process is separated from the loading process, certain information about the translated program must be retained besides the machine language instructions (e.g., the names of subroutines it calls, the location-dependent parts of the program, etc.). For uniformity, a data structure called an object module is defined for this purpose and a single system program called the dynamic loader is used to load programs in object-module form into main memory in executable form. This enables language processors like *FTN and *ASMH to translate programs into object-module form and save them on auxiliary storage. Subsequently, the object program can be loaded into main memory by the system loader and executed without invoking a language processor.

This section describes the facilities that are provided by the system loader in MTS. The loader provides only primitive facilities for editing programs in object-module form. Additional facilities in MTS are provided by the object-file editor and the linkage editor which are described in the sections "The Object-File Editor" and "The Linkage Editor."

Overview of the Loading Process

The loader can be invoked at the MTS command language level in several different ways:

```
$RUN FDname
$RERUN FDname
$LOAD FDname
$DEBUG FDname
```

Invocation by a \$RUN or \$RERUN command:
(1) Memory, files, and devices acquired by previous commands and all of their consequences are released.
(2) Devices required for execution of the program are acquired.
(3) The program is loaded and a map is produced if the MAP parameter was specified.
(4) The registers are set up for a normal subroutine call.
(5) The system transfers to the program at the entry point.
(6) The program can return to the system via normal subroutine return conventions or by calling the system subroutines SYSTEM or ERROR.

Invocation by a \$LOAD command:
This is processed exactly as a \$RUN command, except that the loaded program is not executed; instead, the system remains in MTS command mode. Execution of the program may be started via the $\$ S T A R T$ or \$RESTART command. This allows the user to display and alter sections of the loaded program before execution begins.

Invocation by a \$DEBUG command:
This is processed exactly like the \$LOAD command except that debug command mode is entered and Symbolic Debugging System symbol tables are constructed using the information provided by the object program.

The loader can also be invoked dynamically during execution via calls to the subroutines LINK, LOAD, and XCTL (see the subsection "Dynamic Loading"). In each case, the loader processes one or more object modules contained in one or more files. The files to be accessed may be specified both explicitly and implicitly. For example,
\$RUN MAIN+SUBR+*PL1LIB
In this case, the loader processes each object module in the files MAIN and SUBR. It selectively processes only required object modules from the library file *PL1LIB. If, after processing the explicitly named files, there are still unresolved external symbols (normally subroutine calls), the loader will implicitly search any private user files specified previously by the \$SET LIBSRCH=FDname command. If there are still unresolved symbols, the loader will implicitly search the system library file *LIBRARY and the resident system symbol table. If there are still unresolved symbols, the loader will prompt conversational users for more input; batch users will get an error message from the loader and the loading process will be aborted.

During the loading process, several different types of output will be produced by the loader:
(1) An image of the program is constructed in executable form in the user's virtual memory.
(2) A unique entry point is determined that is the address of the
first instruction to be executed in the loaded program (see Appendix A of this section for the specific algorithm used).
(3) Error messages may be produced if unusual conditions are detected (see Appendix B for a list of error messages).
(4) A map is produced on request specifying the entry point of the program and where each independent section was loaded (see Appendix C for a sample map).
(5) A cross-reference listing is produced on request specifying where in the loaded program each external symbol was referenced.

If there remain undefined external symbols after processing all explicitly and implicitly specified files, the conversational user is prompted with the message

$$
\begin{aligned}
& \text { Enter location of more loader input, "CANCEL", "IGNORE", "USMSG", } \\
& \text { "UXREF", or "MAP": }
\end{aligned}
$$

?

The responses can be:
(1) An FDname, which specifies a file or device from which more loader input can be read.
(2) CANCEL, in which case a return is made to MTS command mode. All programs loaded thus far are in the user's virtual memory and can be executed, if desired, with the \$START command unless the loader was invoked via the subroutines LINK or XCTL.
(3) IGNORE, in which case the loading process is completed without defining the specified undefined symbols.
(4) USMSG, in which case the list of undefined symbols that caused the prompting is printed on the logical I/O unit SERCOM and the user is again prompted.
(5) UXREF, in which case a list of the locations of each reference to the undefined symbols that caused the prompting is printed on SERCOM and the user is again prompted.
(6) MAP, in which case a map of the current loading process is printed on SERCOM and the user is again prompted.

If, during the loading process, an illegal loader record is encountered, the erroneous record is printed out and is treated as an end-of-file.

Structure of Compiler-Generated Object Modules

From the loader's point of view, an object program consists of a series of object modules contained in one or more files. The object modules describe the logical structure of the program to be loaded; namely, the name, the origin, and the length of each independent component, the external subroutines that are called, the actual machine instructions and constants, and a description of the location-dependent parts within the program. An object module has no name or length associated with it. Each object module defines one or more components,
each of which is independent in the sense that there is no fixed relationship between where it and any other program components are loaded. These independent components are called control sections (csects) and have a name, an origin, and a length associated with them. Items within a control section must always be loaded a fixed distance apart as specified by the compiler. Hence, a control section is the smallest relocatable unit in an object program.

Each language processor has its own algorithm for translating source programs into object programs. *FTN produces one object module for each main program and subprogram encountered; each object module defines exactly one control section whose name and length are the same as the translated source program. *PL1 produces one object module for each external procedure encountered; each object module defines many control sections of different lengths whose names are generated from related items in the source programs.

Physically, an object program consists of a sequence of loader records contained in one or more files. Each compiler-generated object module consists of sequence of loader records of the following types:

```
SYM records: Symbolic Debugging System information
ESD records: external symbol dictionary
TXT records: machine text and constants
RLD records: relocation dictionary
END records: object module terminator
```

SYM records are produced by *ASMG, *ASMH, *ASMT, *FTN, *PL1, and *PL360 when the TEST option is specified. The information contained on these records is used by SDS (the Symbolic Debugging System) to construct symbol tables internal to SDS for debugging and is completely ignored by the loader.

ESD records contain the description of each control section in the module (name, origin, and length), a description of alternate entry points within the module, a description of the common sections and pseudo-registers required by the module, and the names of external subroutines called by the control sections in this module.

TXT records contain the actual machine instructions and constants to be inserted in the control sections in the module.

RLD records contain information about which parts of the control sections in this module have location-dependent items which must be modified during loading.

END records serve as object module terminators and can optionally specify the program entry point (see Appendix $A$ on entry point determination).

The ordering restrictions on the loader records in an object module are minimal.
(1) All SYM records (if any) must come first.
(2) The ESD record defining a control section must come before the associated TXT and RLD records.
(3) The TXT record for a location-dependent item must come before the associated RLD record.
(4) The END record terminates the current object module.

Hence, an object module is roughly characterized as beginning with either a SYM or ESD record and terminating with an END record. In between, there can be any convenient combination of record types which satisfy the above restrictions.

Three system utility programs are available for scanning object programs--*OBJSCAN, *OBJLIST, and *OBJUTIL. *OBJSCAN gives the order of the loader records in an object file. *OBJLIST interprets the information on each loader record in an object file. See MTS Volume 2, Public File Descriptions, for the complete descriptions of *OBJSCAN and *OBJLIST. *OBJUTIL is a general-purpose object-file utility program that provides similar capabilities through its command language. *OBJUTIL is described in the section "The Object File Editor" in this volume.

## Processing an Object Module

The loader constructs a loader symbol table of the names of all control sections encountered during the loading process. ESD records are the main source of such names. They contain the names, origins, and lengths of control sections defined in the current module and the names of control sections that are referenced by the current module but defined elsewhere. If a control section definition is encountered, the length is used to allocate space for it in the user's virtual memory and an entry is made in the symbol table with its corresponding virtual memory address and length. If a control section reference is encountered and that control section has not been defined by a previous module, an entry is made in the symbol table with the unresolved flag set. All symbols on ESD records are given an external symbol identification number (ESID) by the compiler (1, 2, 3, etc.) that will be used on subsequent TXT and RLD records to indicate which control sections are involved. These ESIDs are unique only within an object module and need not start with 1 or be sequential.

TXT records contain the text to be inserted in a control section. The ESID of the appropriate control section and the displacement within the control section is taken from the TXT record to compute the virtual memory address at which the text should be inserted.

RLD records specify those text items that are location-dependent and must be modified. Each item on an RLD record specifies the ESID of the control section containing the location-dependent data, its displacement within the control section, its length and alignment, and finally the ESID of the control section on whose location the data item is
dependent. Each item of this type is modified by the loader using this information and the symbol table.

The END record terminates processing of the current object module.
Note that while already complex, this is a greatly simplified view of object-module processing. In particular, the discussion of the following topics are deferred for later subsections:

- multiply-defined control sections
- common and pseudo-register processing
- block data subprograms
- external references
- initial ESD lists and low-core symbol tables

Storage Allocation During the Loading Process


#### Abstract

The loader allocates storage in user virtual memory for each control section and common section encountered during the loading process. This storage is allocated via separate calls to the GETSPACE subroutine for each control section and common section. Hence, no assumptions should be made as to where a particular section will be loaded in user virtual memory. The loader also builds dynamic tables in system virtual memory. The loader table storage is released before execution begins if the SYMTAB option of the \$SET command is OFF; the default is ON, in which case the tables are not released and information about the loaded program is available during execution for use by SDS, the loader, MTS, and user programs via the subroutine LOADINFO. All storage allocated during loading is released when the program terminates normally (i.e., stops with the EXECUTION TERMINATED message). Otherwise, the storage is not released until the \$UNLOAD command or another \$RUN, \$RERUN, \$LOAD, or $\$$ DEBUG command is specified.


ADVANCED LOADER TOPICS

Modifying Programs in Object-Module Form

There are currently three methods for modifying object programs.
The least recommended method is the use of the MTS file editor (\$EDIT) when an object program is in a file. In this case, one can scan for certain strings and replace them. Care must be taken to make one-for-one replacements since loader records have fixed-length fields. It may also happen that the string scanned for is broken across two loader records and not found. The editor is best used to change the names contained on ESD records.

The least error-prone method is to use the facilities of the MTS linkage editor or object file editor which are described in the sections "The Linkage Editor" and "The Object File Editor" in this volume.

A less expensive, but more error-prone, method involves inserting user-generated records into the object file in appropriate places to modify the loading process. This usually requires the object program to be in a line file since records are generally inserted into rather than added to the end of a module. It also requires knowledge of object module itself via *OBJSCAN and *OBJLIST. A brief description of the user-generated records follows. The physical format of these records is specified in the appendices.

## ALI Record (Alias)

The ALI record may be used to specify symbols that are to be treated identically, i.e., as synonyms, during the loading process. The record may be inserted anywhere within an object module with the only restriction that both symbols on the ALI record may not be previously defined (although either one may be). A symbol may have any number of aliases but only one alias may be defined per ALI record.

COM Record (Comment)
The COM record is completely ignored by the loader. It can be used to insert a comment anywhere in an object program.

DEF Record (Define)
The DEF record can be used to define an external symbol currently undefined in the object program. A symbol can be defined in two ways: an absolute definition or a relative definition. An absolute DEF record is independent of any other symbols defined in the object program and can be inserted anywhere. The value of the symbol is its actual virtual memory address. A relative DEF record defines a symbol relative to an existing control section in the object program. Relative DEF records must appear in the module which defines the associated control section. In this case, the value is specified by giving the ESID of the associated control section and a relative address within it.

ENT Record (Entry)
An ENT record may be used to specify the entry point of the object program. The record contains the external name of the first instruction to be executed and may be inserted anywhere in the object program. See Appendix A for a complete description of entry point determination.

LCS Record (Low Core Symbol)
An LCS record may be inserted before or after an object module to force the loader to scan a low-core symbol table (previously defined) for definitions or unresolved symbols. See the section describing initial ESD lists and low-core symbol tables.

LDT Record (Load Terminate)
An LDT record can be used as a pseudo-end-of-file for the loader. The loader will never read past an LDT record; hence, it is useful if other information besides an object program is to be stored in a file. It can also be used to specify the object program entry point.

MDL Record (쓴ultiple Definition Loading)
The MDL record controls the handing of multiply-defined control sections and must be inserted before or after object modules. Normally, all occurrences of a control section except the first are ignored. The MDL ON record specifies that subsequent occurrences of a control section are to be overlayed on the first. This requires that subsequent definitions of a control section be no longer than the first and that address constants be defined by only one occurrence of the control section or else be defined the same in all occurrences; otherwise, unpredictable results may occur.

MSG Record (Message)
The MSG record may be used to produce a message on the map file or device (defaults to *SINK*) at the point at which the MSG record is encountered by the loader.

NCA Record (No Care)
The NCA record may be used to specify that the user does not care if a symbol is undefined. The NCA record has effect only if it precedes all references to the symbol specified. Any symbol left undefined by an NCA record has the value zero; hence, a program can test whether a symbol is undefined. If an NCA record is the first reference to a symbol, it has the following effects:
(1) The loader makes no attempt to resolve the symbol when searching libraries and low-core symbol tables.
(2) The loader does not consider the symbol undefined if it has not been defined by the end of the loading process.

OPT Record (Options)
The OPT record may be used to control various options during the loading process. Currently, the following options are allowed:
(1) $M D L=\{O N \mid O F F\}$

The MDL option is equivalent to the MDL loader record function described above.
(2) WARN= $\{O N \mid O F F\}$

Nonfatal loader error messages are printed if WARN is set to ON (the default); otherwise, these error messages are suppressed.
(3) NULMSG=\{ON|OFF\}

The NULMSG option controls the action of the loader when an attempt is made to load a null program. If NULMSG is ON (the default), the loader prints an error message and, if in conversational mode, prompts the user for more loader input; for batch mode, the loading operation is aborted. If NULMSG is OFF, the error message is suppressed and the user is not prompted for more input, i.e., the loading operation is aborted (this is the same as if the user had entered "CANCEL" in response to a request for more loader input).
(4) NCALCS= $\{O N \mid O F F\}$

If the NCALCS option is ON, the loader will attempt to resolve symbols defined by NCA records when searching the low-core symbol tables. If NCALCS is OFF (the default), no attempt is made to resolve these symbols.
(5) PUSH

The PUSH option saves the current status of the OPT record options.
(6) POP

The POP option restores the OPT record options to the previous PUSH status.
(7) $F I L L=\{x x \mid O F F\}$

If "xx" is specified, the loader will preset the storage allocated for each control section to the fill string "xx", before any associated text is processed. "xx" may be from 2 to 16 hexadecimal digits of appropriate length, i.e., byte, halfword, fullword, or doubleword. If OFF is specified, the loader will not fill memory (the default).
(8) $\operatorname{SAVESD}=\{O N \mid O F F\}$

By default, the loader will purge any symbols from its symbol tables that have not been referenced by any RLD (relocation
dictionary) items or RIP records. This action corresponds to the option SAVESD=OFF. If SAVESD=ON is specified, all external symbols will be marked as referenced, forcing the loader to attempt to resolve these "unreferenced" symbols.
(9) $W X L C S=\{O N \mid O F F\}$

If the WXLCS option is ON, the loader will attempt to resolve symbols of type $W X$ (weak external reference) when searching the low-core symbol tables. If WXLCS is OFF (the default) no attempt is made to resolve these symbols.
(10)
$\mathrm{BTCRPL}=\{\mathrm{ON} \mid \underline{\mathrm{OFF}}\}$
The BTCRPL (branch-type constant replace) option controls the loader relocation of branch-type address constants (V-adcon) when RLD records are processed. If BTCRPL=OFF is specified (the default), the loader adds the value of the $V$-adcon to the contents of the location specified by the RLD item address. If BTCRPL=ON is specified, the loader replaces the contents of the location specified by the corresponding RLD item with the V -adcon. This option is normally not required since most MTS compilers produce TXT or CSI records that have zero values in the locations that have branch-type adcon contants.
(11)

DEFPRV
The DEFPRV option forces the loader to resolve all pseudoregister references that have been encountered up to this point in the load. Any currently undefined pseudo-register references are defined, any undefined CXDs are set to the current cumulative PRV length, and the cumulative PRV length is reset to 0. Subsequent pseudo-register references are resolved as follows:
(a) if the pseudo-register name is the same as a previously resolved pseudo-register, the offset for the new pseudoregister is made the same as the previous occurrence of that pseudo-register.
(b) if this portion of the load refers to any pseudo-registers which were previously resolved, any new pseudo-registers will be given offsets greater than any previously defined pseudo-registers. Otherwise, the offsets for any new pseudo-registers begin at 0 .

Any number of options may be specified on the OPT record. Each option must be separated by a comma. The processing of the option string is terminated by the first blank or by a syntax error.

REP Records (Replace)
The REP record can be used to correct erroneous text in a control section by overlaying the original text. This means a REP record
must be inserted into an object module after the compiler-generated TXT record containing the original text and must be a one-for-one replacement. If it corrects a location-dependent item, it must be inserted before the compiler-generated RLD record specifying that item. In all cases, REP records must be inserted before the module END record. The replacement text is a variable-length sequence of pairs of hexadecimal numbers which can be separated by commas for readability, and which is terminated by a blank. The text being replaced can begin on any byte boundary.

RIP Record (Reference If Present)
The RIP record has two uses. The primary use is in object module libraries (see the subsection "Library Processing"). It can also be used to generate an unconditional reference to a symbol which specifies to the loader that the symbol in question must be defined before the loading process terminates. This is a convenient way to force a symbol to be defined during loading even though there are no compiler-generated references to the symbol (see, for example, the use of unconditional RIP records in the subsection "Initial

MTS 5: System Services

ESD-Lists and Low-Core Symbol Tables). RIP records must be inserted before or after object modules.

## Common-Section Processing

In addition to defining control sections, an object module may define common sections. Common sections are similar to control sections in that both have names, origins, and lengths associated with them. Common sections differ from control sections in that they may have no text associated with them (and hence no RLD items) and any object module in an object program may redefine a common section with a different origin and length. It is the job of the loader to collect these various definitions, allocate storage for it using the longest length encountered, and resolve individual object module references to the common sections using the appropriate origin. This means that storage allocation for common sections is deferred until all object modules have been processed.

Since no text may be associated with common sections, the problem of preinitializing common sections arises. To solve this problem, control sections having the same name as common sections are merged. Since control sections can have text in them, overlaying a control section on a common section has the effect of initializing it. This is how the FORTRAN BLOCK DATA subprogram works. It should be noted however that when a control section of this type is encountered, the loader must allocate storage for the common area at that time since text may now be inserted. This means that the control section must specify the longest length required for the common area or that the longest length has already been encountered.

## Pseudo-Register Processing

In addition to defining control sections and common sections, an object module may define pseudo-register areas. The name pseudoregister comes from a particular application of this facility in PL/I object programs. The concept itself is better described in assembler language terms as a global or external dsect (external dummy control section). It is a dsect in the sense that no storage is allocated for it by the loader. It is the program's responsibility to allocate storage and establish addressability. It is external or global in the sense that there can be only one external dsect. It has no name; however, the name of each element in it is an external symbol. Object modules define elements in the external dsect (pseudo-register area) by specifying a name, length, and alignment attribute. The loader collects requests for such elements during the loading process. If more than one specification for a particular element is encountered, the longest length and strictest alignment are retained. Finally, when all object
modules have been processed, the loader assigns to each item a displacement within the external dsect according to the accumulated length and alignment attributes (just like the assembler does for internal dsects). The value of symbols in the external dsect area will be their offset within the area. The loader also computes the cumulative length of the area. The total length can be accessed by the object program via a special RLD item and used to allocate the storage for the pseudo-register area at execution time.

## Merging External Symbols

During the loading process the loader builds a table of the external symbols encountered. External symbols are by definition 8 characters long, padded on the right with blanks if necessary. External symbols can be the names of control sections, common sections, pseudo-register area items, entry points within control sections, predefined symbols, or references to external symbols defined elsewhere. The loader must integrate these external symbols as they are encountered during the loading process in order to resolve symbolic references between various components of the object program. The conventions that are used have evolved over the years to accommodate the various language processors.
(1) Control sections with the same name are considered multiplydefined. All occurrences except the first are ignored unless MDL is ON (see the discussion of the MDL record).
(2) Common sections with the same name are identical and are overlayed with control sections of the same name.
(3) There is a special type of control section called a private control section (psect) which has no name (8 blanks). Each occurrence of a psect is unique and independent of any others. Moreover, psects do not merge with blank-named common sections. This is why no block-data subprograms are allowed for blanknamed common sections.
(4) Pseudo-register area items with the same name are identical. Such items can have the same name as control sections and common sections, but are considered unique and independent.

The following table summarizes the action taken by the loader when it encounters a symbol in the input stream which is already in the loader table:

Symbol Type in Loader Tables

|  |  | CSECT | COMMON | PR | PSECT | Entry <br> Point | Pre | External Ref. |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  | CSECT | MD | ME | UN | - | MD | MD | ME |
| Symbol | COMIMON | ME | ME | UN | UN | ME | ME | ME |
| Type in | PR | UN | UN | ME | UN | UN | UN | UN |
| Input | PSECT | - | UN | UN | UN | - | UN | - |
| Stream | Entry | MD | ME | UN | - | MD | MD | ME |
|  | Pre-def | MD | ME | UN | UN | MD | MD | ME |
|  | Ex. Ref | ME | ME | UN | - | ME | ME | ME |
| match is not possible |  |  |  |  |  |  |  |  |
| MD: considered a multiple-definition; no new entry is madeME: considered identical and merged |  |  |  |  |  |  |  |  |
|  | consi | dered | que; an | inde | nt en | y is | nta | for both |

## Library Processing

A library of object modules is structured so that the loader can selectively load only those modules that are needed to define undefined symbols. A symbol is considered undefined by the loader if it has been referenced but not defined by the modules already loaded before an NCA record for that symbol was encountered. The basic idea is to abstract from each module in the library a brief description of what symbols each module defines. A library structure consists of additional loader records that contain the library module descriptions. The loader looks at these records, and processes the associated module only if it resolves an outstanding reference. Currently, there are three types of library control records:

LIB Record (Library)
The LIB record immediately precedes the first record of its associated object module. It contains the name of the first nonblank control section or entry point in the associated object module and may optionally contain a pointer to the next item in the library. The loader processes the LIB record by checking the name on the LIB record. If the name matches an undefined symbol in the loader tables, the module is processed. If not, the module is ignored by using the pointer to the next library item (if available) or by reading and ignoring each record in the module to be skipped.

RIP record (Reference If Present)
The RIP record is used in conjunction with LIB records to solve two problems: multiple entry point library modules and multiple passes over the library. RIP records force the loader to put additional undefined symbols in the loader table conditional upon which symbols are currently undefined. They take the form

RIP SYM1 SYM2
and specify that if SYM2 is currently undefined, then, if SYM1 is not already defined, it should be inserted in the loader tables as undefined.

The multiple entry point problem arises when a library module defines more than one symbol. Since its LIB record can contain only one symbol, some provision must be made for references to the other entry points. This is solved by preceding the LIB record with one RIP record for each alternate entry point in the module as follows:


Any unresolved references to the alternate entry points A1, A2, ... will force an unresolved reference to M1 (unless already present) and hence the module defining M1, A1, A2, ... will be processed when the LIB record is encountered.

The problem of multiple library scans arises when modules in a library reference each other. In general, it is impossible to order the modules so that there are no back references to preceding modules. Hence, the loader may ignore library module A because there were no unresolved references to $A$ when it was encountered even thought a subsequent library module required loading and references A. Since it would be expensive and inefficient to rescan the library until all undefined symbols have been resolved, the problem of back references is solved by beginning the library with a sequence of RIP records, one for each back reference as follows:


In this case an unresolved reference to B will force an unresolved reference to A (unless already present). Consequently, both A and $B$ will be loaded in one pass.

DIR Record (Directory)
The DIR record is used to collect the entry point information for all modules in the library into one large directory at the beginning of the library. The DIR record contains the size of the directory and a pointer to the end of the library. The directory itself is the very next record after the DIR record. It contains one entry for each symbol defined in the library and a corresponding pointer to the module which defines that symbol. The loader scans the directory for matches to undefined symbols and uses the associated pointer to process the defining module. When a complete pass over the directory is made with no matches, the loader skips over the library using the end-of-library pointer.

Libraries are generated in MTS via the program *OBJUTIL. *OBJUTIL currently generates DIR-type libraries although there are three other types of libraries: sequential, line, and POINT.

Sequential Libraries
Sequential libraries are the most primitive and inefficient form of library available. They consist of a sequence of modules preceded by their library control records. No information is retained which allows the loader to efficiently skip a module not required. A sequential library can be stored in either a line file or a sequential file.

Line Libraries
A more efficient library format can be obtained in a line file by separating the modules from their library control records. Since, by default, reading begins a line 1, the modules are stored in the negative line number range and the control records in the positive range. In this case, the control records contain the (negative) line number of the associated modules which the loader uses if the module is required. Otherwise, the module is skipped by reading the next control record. The line library is reasonably efficient and is the easiest type to edit using the MTS file editor.

POINT Libraries
A similar improvement in efficiency can be obtained in sequential files by using the POINT format. In this case, the library consists of a sequence of modules together with their library control records. However, NOTE-POINT information is contained in the control records which allows the loader to skip over modules which are not required. A POINT library is difficult to edit because it must be in a sequential file.

## DIR Libraries

The most efficient library structure is obtained in line or sequential files by using the DIR format. In this case, all library control information is contained in one directory at the beginning of the file. The loader uses the directory information to process the required modules. The efficiency of the DIR format stems from the minimal number of I/O operations required. *LIBRARY is an example of a DIR library.

When generating a library, the recommended approach is to use the DIR format in a line file initially to allow easy editing via *OBJUTIL. When the library modules stabilize, they should be optimized via *LINKEDIT or *OBJUTIL and included in a DIR library for maximum loading efficiency.

## Dynamic Loading

The ability to dynamically load, execute, and unload programs during execution is provided by the subroutines LINK, LOAD, XCTL, and UNLOAD. Because of the rate structure used in MTS, the cost of running large programs over a period of time can be reduced by minimizing the storage integral via the dynamic loading and unloading of program segments. This cost reduction must be balanced against the additional cost incurred by the increased number of files that must be opened and closed and the increased number of library searches required. In general, the net savings warrants dynamic loading only if the storage integral can be substantially reduced.

A routine can be dynamically loaded by calling one of three subroutines: LINK, LOAD, or XCTL. Each one provides a unique control facility as described below:

Dynamic loading via the subroutine LINK:
(1) The program is loaded and a map is produced if the MAP parameter was specified on the initial \$RUN, \$RERUN, \$DEBUG, or \$LOAD command.
(2) The registers are set up to make the call upon the LINK subroutine transparent.
(3) The system transfers to the program at the entry point.
(4) If the program returns as a normal subroutine, all storage acquired to load it plus all further storage acquired by it at its storage level and at higher levels is released.
(5) The system returns to the program that called LINK.

Dynamic loading via the subroutine LOAD:
(1) The program is loaded and a map is produced if the MAP parameter was specified on the initial \$RUN, \$RERUN, \$DEBUG, or \$LOAD command.
(2) The external symbol table and entry point of the loaded program are made available to the program that called LOAD.
(3) The system returns to the program that called LOAD.

Note that the program loaded is in the user's virtual memory and is ready to use. All storage acquired during the loading operation is catalogued under a unique storage index number. This storage can later be released by calling the subroutine UNLOAD.

Dynamic loading via the subroutine XCTL:
(1) The storage having the current storage index number is released. This presumably is the program which is calling XCTL.
(2) The program is loaded and a map is produced if the MAP parameter was specified on the initial \$RUN, \$RERUN, \$DEBUG, or \$LOAD command.
(3) The registers are set up to make the call upon the XCTL subroutine transparent. It is necessary that the program calling XCTL restore the general registers, including the save area register (13), to what they were at the time it was called (presumably due to a \$RUN, \$RERUN, \$DEBUG, or \$LOAD command or a call on LINK or XCTL). The user may have XCTL perform this function by specifying the appropriate option when calling XCTL.
(4) The system transfers to the program at the entry point.
(5) If the program returns as a normal subroutine, the action taken is the same as that which would have occurred if the program that called XCTL had returned instead.

How the program being dynamically loaded interacts with the currently loaded programs depends on several things. If SYMTAB is OFF, no information about the currently loaded programs is available during
execution. Hence, the dynamically loaded program is loaded independently of the currently loaded programs unless special features like initial ESD lists or low-core symbol tables are used (see the following subsection). Consequently, more than one occurrence of a library subroutine may be loaded. If, however, SYMTAB is ON, information about the currently loaded programs is available during execution and can be used to merge the program being loaded with the currently loaded programs. This is controlled by the MERGE bit which must be specified when calling LINK, LOAD, or XCTL.

The merging process occurs in both directions. Undefined symbols from previous loads are resolved if possible by the program being currently loaded and undefined symbols in the program being loaded are resolved if possible by programs already loaded. Common sections and block data subprograms are also merged with those previously loaded if the merge bit is set.

Each time LINK, LOAD, or XCTL is invoked, a unique storage index number is defined and associated with the storage allocated during the loading process (see the section "Virtual Memory Management" in this volume). This storage index number is used by the loader to distinguish which symbols were defined during a particular loading process. Hence, it is permissible to "multiply define" a symbol during dynamic loading which was defined during a previous loading process. Thus, for example, a routine named A can dynamically load itself or any other routine named A. Note, however, that it is still not permissible to load two definitions of A during the same loading process.

The storage acquired during dynamic loading is automatically released when the loaded program returns to LINK or XCTL. It is the caller's responsibility, however, to release storage acquired via a call to LOAD. This is accomplished by calling the UNLOAD subroutine. Only routines dynamically loaded via the LOAD subroutine may be unloaded in this fashion. Moreover, it is not possible to unload only part of a dynamically loaded program; all storage acquired during a particular call to LOAD is released by UNLOAD. An exception to this rule can occur if LOAD has been called several times when SYMTAB is ON and the merge bit is set. For example, suppose on the first call to LOAD, a FORTRAN routine is loaded which causes FREAD to be loaded from *LIBRARY. Suppose a second call to LOAD loads another FORTRAN routine which references FREAD. If SYMTAB is ON and the merge bit is set, a second copy of FREAD will not be loaded. If UNLOAD is now called to release the first routine, FREAD will not be released even though it was loaded at the same time. Rather, it will be unloaded when the last routine referencing it is unloaded.

Initial ESD Lists and Low-Core Symbol Tables

The dynamic loader allows external symbols to be predefined at the beginning of the loading process. It will also search an "external
symbol dictionary" for the definitions of external symbols which have been referred to, but not yet defined, whenever an LCS record is encountered. Both of these facilities are available through the LINK, LOAD, and XCTL subroutines. They are a convenient way to allow the programs loaded via LINK, LOAD, and XCTL to refer to routines or data items that are already loaded when the SYMTAB option is OFF.

Both the predefined symbol and "low-core symbol" capabilities require tables called external symbol dictionaries. An external symbol dictionary consists of ( $12 * \mathrm{n}+4$ ) bytes, where "n" is the number of external symbols in the dictionary. The dictionary must begin on a fullword boundary. The first word of the dictionary contains "n", the integer number of symbols entered in the dictionary. The rest of the table consists of external symbol entries, each being twelve bytes long. The first eight bytes of a symbol entry contain the EBCDIC name of the symbol, left-justified with trailing blanks. The last four bytes of the entry contain the value of the symbol as a fullword integer. If the value field is zero, the entry is interpreted as a reference to that symbol rather than a definition.


As an example, suppose that a language scanner is to be broken into an overlay structure consisting of a scanner which links to a different module for each type of statement. Each statement module may in turn refer to the subroutines EXP, PUNT, GEN, and NXTCHAR to scan the next expression, print on error message, generate output, and acquire the next character, respectively. An external symbol dictionary to define these symbols might be:

| MYEXTSYM | DC | $F^{\prime} 5^{\prime}$ | Five Entries |
| :---: | :---: | :---: | :---: |
|  | DC | CL8'EXP' | Defines EXP |
|  | DC | A (EXPRTN) | Internal, different name |
|  | DC | CL8'PUNT' | Define PUNT |
|  | DC | A (PUNT) | Internal, same name |
|  | DC | CL8'GEN' | Define GEN |
|  | DC | V (GENRTN) | External, different name |
|  | DC | CL8' ${ }^{\text {NCTCHAR }}$ | Define NCTCHAR |
|  | DC | V (NXTCHAR) | External, same name |
|  | DC | CL8' PLOT1' | Reference PLOT1 so |
|  | DC | A (0) | it will be loaded. |

Notice that it matters not whether the symbol is external or internal in the routines already loaded. It may even have a different name in the program already loaded.

Now these symbols and the definitions given above might be used as an initial ESD list when calling LINK, LOAD, or XCTL. The symbols in the above table would then be appended to the symbol which MTS normally predefines (LCSYMBOL) and would become defined (or referenced) during the initialization of the loader. These symbols can then be referred to by the object modules being loaded. No object module can redefine any predefined symbols, however. So these external symbols are reserved symbols to the program being loaded.

The external symbol table constructed above (with the last entry deleted) could alternatively be used as a "low-core" symbol dictionary. The loader would search the external symbol dictionary for the definitions of external symbols which have been referred to, but not yet defined, whenever an LCS record referring to the low-core symbol dictionary was encountered. The advantage of this method is that the program being loaded can define its own symbols with the same names as some of the symbols in the low-core symbol dictionary without conflict; the program's own definition will be preferred.

In order to refer to the low-core symbol dictionary, the loader must have an external symbol defined that is the base of the table. This can be accomplished by making the name of the table a predefined external symbol. For instance, in our example above we might call LINK, LOAD, or XCTL with the following initial ESD list:

```
PREDEF DC F'1' One entry
    DC CL8'MYEXTSYM' Define MYEXTSYM
    DC A (MYEXTSYM)
```

Then the occurrence of an LCS record referring to the symbol MYEXTSYM will cause each symbol contained in that external symbol dictionary to be defined with the value given in the dictionary if it has been referred to, but not defined, by the object modules loaded thus far. If the object modules loaded define any of the symbols in the table, the definition given by the modules will override the dictionary definition.

Pseudo-register areas may be defined via low-core symbol tables that have a slightly different structure:


The first 8 bytes give the number of entries and the cumulative length for the pseudo-register area. The entries for the pseudo-register areas

May 1983
follow. Each entry is 16 bytes long rather than 12 bytes and specifies both length and alignment of each area. The address entry is 4 bytes. The length entry is 3 bytes and the length is given in bytes. The alignment entry is 1 byte and the alignment codes are

$$
\begin{aligned}
& X^{\prime} 00^{\prime}: \\
& X^{\prime} 01^{\prime}: \\
& X^{\prime} 03^{\prime}: \text { byte aligword alignment } \\
& X^{\prime} 07^{\prime}: \\
& \text { fullword alignment } \\
& \text { doubleword alignment }
\end{aligned}
$$

Low-core pseudo-register tables are distinguished by the characters PR occurring on all LCS records which refer to them (see Appendix D for the exact LCS record format).

It should be pointed out that the symbols defined as predefined symbols or in low-core symbol dictionaries may have as their values the addresses of any items in the loaded program that the user wants to refer to in the program to be loaded. For instance, the address of a subroutine, a common section, or a variable might be passed on. Furthermore, the address given as the definition of an external symbol need not have the same name or even any name in the already loaded program. It is just an address which is associated with EBCDIC characters in the table to form an external symbol definition for the program to be loaded.

There are currently four low-core symbol tables that define user entry points into the resident system: LCSYMBOL, <EFL>, <FIX>, and PL1SYM. LCSYMBOL is always predefined. Hence, the record

LCS LCSYMBOL
can be inserted into an object file whenever desired. However, in order to reference <EFL>, <FIX>, or PL1SYM, they must first be defined for the loader. Their definitions are entries in LCSYMBOL and can be extracted as follows:

| RIP | <EFL> |  |
| :--- | :--- | :--- |
| RIP | <FIX> |  |
| RIP | PL1SYM |  |
| LCS |  | LCSYMBOL |
| LCS |  | <EFL> |
| LCS |  | <FIX> |
| LCS |  | PLISYM |

## APPENDIX A: ENTRY POINT DETERMINATION

The entry point of a program is the point within the program where execution will begin if the program is executed. That is, it is the location of the first instruction within the program to be executed. This location is determined by the loader as it processes the object modules in the object program. The following algorithm is used to determine the entry point:
(1) If one or more entry (ENT) records have been encountered, the entry point is the value of the external symbol referred to on the first such ENT record.
(2) Otherwise, if the processing of input from the user-specified source was terminated by a load-terminate (LDT) record and that record referred to an external symbol, the entry point is the value of that external symbol.
(3) Otherwise, if one or more object module end (END) records contain an entry point address or external symbol, the entry point is the value of the entry point address or the address of the external symbol from the first such END record.
(4) Otherwise, if at least one control section has been encountered, the entry point is the address of the first byte of the first control section encountered.
(5) Otherwise, the entry point is zero (undefined).

Note: The 360/370-assembler puts an entry address on the END record if one was given on the END statement by the programmer. FORTRAN puts an entry address on the END record for main programs but not for subroutines. PL/I puts an entry address on the END record of each program.

## APPENDIX B: LOADER ERROR MESSAGES

As the loader is processing its input, it checks the input for errors. An error message is printed for each error detected and the action taken next depends upon the severity of the error. If the error is a nonrecoverable error, loading is terminated immediately and execution of the program is aborted. If the error is is a fatal error, loading is continued but the execution of the program is still aborted. Finally, if the error is a nonfatal error, loading continues as normal. A list showing each message, its severity, and likely causes for the message is given below:

ABNORMAL LOADER INPUT ORDERING. (xxx-yyy)
Nonfatal. The loader input record type "xxx" is followed by the loader input record type "Yyy" which is considered abnormal. The loader will attempt to process the records in this order, but further error comments may appear if this is not possible. This error usually is caused by failure to empty an object file before recompiling a program.

ADDRESS OUTSIDE OF CONTROL SECTION xxxxxxxx BOUNDS.
Nonfatal. Some reference to an address within a control section is made with a relocatable address which does not fall within the address range of the control section (the starting address to the starting address plus the length). The data which was to be moved to that address or the relocation which was to be performed at that address is ignored. The usual reasons for this error are an incorrect $R E P$ record or the second occurrence of a control section which is longer than the first.

AN OCCURRENCE OF xxxxxxxx LONGER THAN FIRST.
Nonfatal. A subsequent occurrence of control section "xxxxxxxx" is longer than the first. Subsequent occurrences are ignored completely in loading unless an MDL ON record is in effect.

AT LINE xxxxx.xxx
This message is printed out in conjunction with other error messages to help locate the error. The "xxxxx.xxx" is the line number of the current input record.

BOTH xxxxxxxx AND yyyyyyyy ARE ALREADY DEFINED. ALI CARD IGNORED.
Nonfatal. Both symbols specified on an ALI record have been previously defined. The ALI record is ignored.

CARD NOT A LOAD CARD, END-OF-FILE ASSUMED.
Nonfatal. An input record which did not have a legal loader record type in columns 2-4 was read. The contents of the record are printed, and are lost for further processing. The loader continues as if this had been an end-of-file.

ERROR WHILE PROCESSING LIBRARY DIRECTORY.
Fatal. The DIR record and its associated directory are unable to be processed.

ERROR: THE LOW CORE AND MODULES SPECS CONFLICT FOR PR xxxxxxxx.
Nonfatal. A module reference to a pseudo-register area, which has previously been defined in a low-core symbol table, specifies a longer length or a stricter alignment.

ESID xxxx RELOCATION FACTOR UNDEFINED.
Fatal. Information on an ESD, TXT, REP, DEF, RLD, or END record is to be relocated relative to the symbol which has the specified ESID. However, either no symbol has that ESID, or the symbol which does have that ESID is not the name of a control section and hence does not have a relocation factor assigned with it. Common causes of this error are
(a) Mixing up the order of the object module records.
(b) Leaving object module records in a file which contains a newer, but shorter object module so that the loader attempts to process these old, obsolete records.

ESID xxxx UNDEFINED.
Fatal. An address constant or channel command word refers to the symbol with the specified ESID within its expression. However, no external symbol has been given this ESID in the object module being processed. This message is caused by the same errors as listed in the error message above.

FORWARD REFERENCING ENTRY POINTS ARE UNDEFINED.
Nonfatal. The control section required to define one or more entry points was not found. The entry points are ignored and processing continues. This is usually caused by a scrambled object file.

GETSPACE UNABLE TO OBTAIN SPACE.
Fatal. The GETSPACE subroutine returned a nonzero return code when called by the loader to allocate storage for a control section or common definition. This error may be caused by an invalid ESD record, or a control section or common definition
that requires more than one segment of storage (e.g., a FORTRAN array that was declared larger than 1,048,576 bytes). This error also may be result of attempting to load a program that requires more storage than is available for user programs.

ILLEGAL HEX CHARACTER DETECTED.
Nonfatal. An illegal hex character was detected on a usergenerated REP or DEF record. The record is ignored and processing continues.

ILLEGAL OPTION OR SYNTAX ERROR ON "OPT" CARD.

Nonfatal. An illegal option or syntax error was encountered on an OPT record.

ILLEGAL TEXT CARD LENGTH DETECTED.

Nonfatal. A text record was encountered with a nonpositive length. The text record is ignored and processing continues.

IN THE MODULE WHICH DEFINES nnnnnnnn.

This message is printed out in conjunction with other error messages to help locate the error. "nnnnnnnn" is the name of one of the control sections in the current object module.

INPUT RECORD LONGER THAN 256 CHARS.
Fatal. An input record was read which was longer than 256 characters.

LOADING ERRORS ABOVE ARE NON-RECOVERABLE.
This message should never appear, except perhaps when trying to force a map.

MISSING END CARD DETECTED.
Nonfatal. An end-of-file was encountered while processing an object module before the associated END record was processed. This is usually caused by specifying parts of object programs with erroneous line number ranges.

NON-ZERO RETURN CODE FROM 'POINT'.
Nonfatal. The POINT subroutine was called by the loader to skip over an unwanted object module in a library. POINT returned a nonzero return code.

THERE ARE nnn UNDEFINED SYMBOLS.
Fatal. The specified number of symbols have been referred to by at least one of the object modules loaded, but have not been defined. In conversational mode, MTS will prompt the user for more loader input if this is the only fatal error occurring.

THE SYMBOL xxxxxxxx HAS AN ILLEGAL ESD TYPE.
Nonfatal. An item on an ESD record specifies an illegal type. The item is ignored by the loader and processing continues. This error is usually caused by modifying an ESD record with the context editor and not making a one-for-one replacement.

TOO MANY PUSHES OR POPS ON "OPT" CARD.
Nonfatal. More POP options than PUSH options or more than five PUSH options without an intervening POP option were encountered.

UNRESOLVABLE REFERENCE TO xxxxxxxx AT LOCATION nnnnnn BY CSECT yyyyyyyy.

Nonfatal. An RLD item in the module defining csect "yyyyyyyy" could not be processed because the referenced symbol "xxxxxxxx" did not have a relocation factor. This is usually caused by a scrambled object file.

WARNING: A TRUNCATED CSI RECORD DETECTED.
Nonfatal. This is printed when the actual length of the CSI text is shorter than specified and the last character of the text is nonblank. If the last character is a blank, the text is assumed to have been trimmed and is padded with the appropriate number of blanks.

WARNING: CSI RECORD IGNORED.
Nonfatal. The text associated with a CSI record was not loaded. This occurs when the address specified on the CSI record is outside the control section bounds, or when there is not enough space remaining in the control section to hold the associated text.

WARNING: CSI TEXT OVERFLOW DETECTED.
Nonfatal. The actual length of CSI text read was longer than the specified length on the CSI record.

WARNING: LOW CORE SYMBOL TABLE xxxxxxxx HAS AN INVALID ENTRY POINT.
Nonfatal. The first word of the low-core symbol table given in an LCS record is negative or large enough to cause an addressing exception if the table were searched. The LCS record is ignored.

This error usually is caused by specifying a symbol on an LCS record that is not a valid low-core symbol table (e.g., a section definition).

WARNING: LOW CORE SYMBOL TABLE xxxxxxxx IS UNDEFINED.
Nonfatal. The symbol given in an LCS record is currently undefined so that the associated low-core symbol table cannot be referenced.
xxxxxxxx IS MULTIPLY DEFINED. FIRST DEFINITION USED.
Nonfatal. Two object modules containing the definition of the given external symbol have been loaded. The first definition of the symbol is the one actually used in all references to the symbol. The second definition is ignored entirely in loading.

## APPENDIX C: SAMPLE LOADER MAP

A loader map can be generated in several different situations. An explicit request for a map can be made on the \$RUN, \$RERUN, \$LOAD, or \$DEBUG commands, e.g.,
\$RUN MYPROG MAP=-M SCARDS=DATA
\$LOAD MYPROG MAP

In this situation, a map is printed on the device specified (default is *SINK*) during the loading of MYPROG. If MYPROG performs any dynamic loading, a map is also produced on the same device for each call to LINK, LOAD, or XCTL.

If SYMTAB is ON, a map of the currently loaded program can be produced at any time. It can be generated during execution via a call to the subroutine LODMAP. It can be generated when a program terminates abnormally and no map was initially requested on the \$RUN, \$RERUN, \$DEBUG, or \$LOAD command (see the MTS command \$SET ERRMAP=\{ON|OFF\}). Finally, a map can be generated at the MTS command language level via
\$DISPLAY [ON FDname] MAP
A loader map consists of the following items:
Map Delimiters
A map begins and ends with a dotted line delimiter. The printing of the delimiters may be controlled by the MTS command \$SET MAPDOTS=\{ON|OFF\} (default is ON).

Entry Point
The address of the first instruction to be executed as computed by the loader is printed in every map.

Size
The amount of virtual memory (in bytes) allocated for control sections and common sections during a particular loading process is printed in every map.

Pseudo-Register Map
The displacement assigned by the loader to each external symbol in the pseudo-register area is printed along with the length and storage index number (SI\#) associated with each symbol. The printing of this section may be controlled by the MTS command \$SET PRMAP=\{ON|OFF\} (default is OFF).

## Predefined Map

The values of each external symbol defined via an initial ESD list, a low-core symbol table, or an absolute DEF record is printed along with the associated storage index numbers. The printing of this section may be controlled by the MTS command \$SET PDMAP = \{ON|OFF\} (default is OFF).

## Section Map

The name of each common section, control section, and entry point within each control section is printed along with the associated address (value) and storage index number; the relocation factor and lengths are also printed for each common and control section. In addition, a flag field and a type field are printed.

The storage index numbers associated with each symbol in the map are of interest only when the dynamic loading features are being used.

The relocation factor associated with each common and control section in the map is generally of use only to assembly language programmers. The relocation factor is that virtual memory address which must be added to the translator-assigned address of an item in a common or control section in order to obtain the actual virtual memory address at which the item was loaded. The relocation factor can be used to facilitate the debugging of assembly language programs both in MTS command mode and in MTS debug mode as follows:

By issuing the MTS or debug command \$SET RF=nnnnnn, where "nnnnnn" is the loader-assigned relocation factor of a common or control section, items within that section may be displayed via their translator-assigned address rather than their virtual memory address. For example, \$DISPLAY 30 is equivalent to \$DISPLAY 803030 if the command $\$$ SET RF=803000 has been issued.

The relocation factor is obtained by subtracting the translatorassigned origin of a section from its loader-assigned virtual memory address. Hence, the relocation factor will differ from the value only if the section has a nonzero origin. The algorithm for assigning section origins is translator-dependent. *FTN, for example, assigns a zero origin to every section, while *ASMH assigns origins as if each section were to be loaded contiguously beginning at a user-specified origin.

The flag (F) value associated with each common and control section is a collection of bit flags that describe the addressing and relocation modes of the section. The following values are ORed together to produce the flag field:

```
MTS 5: System Services
```

```
        X'08' - Read-only csect (RSECT) attribute
        X'04' - Resident-mode csect attribute
                                (0=RMODE 24, 1=RMODE ANY)
        X'03' - Addressing-mode attribute (2 bits)
        B'00' - AMODE 24
        B'01' - AMODE 24
        B'10' - AMODE 31
        B'11' - AMODE ANY
        The codes for the type (T) field are as follows:
        (blank) - Control section loaded from a normal object file
        L - Control section loaded from a library file
        E - Entry point within a control section
        C - Common section
    The following example illustrates both a PL/I and FORTRAN object
program map as produced by the loader:
```



MTS 5: System Services

```
APPENDIX D: LOADER RECORD FORMAT
```

The following are the card image formats for the dynamic loader. The loader will accept variable-length input up to 256 bytes long, but card images are the most common and hence are shown here. The record following a CSI or DIR record may be up to 32767 bytes long.

Translator-Generated Load Records

ESD Input Record (Card Image)


```
ESD Data Item
```



```
                    PC, or CM
                            -Identifier of SD entry containing the entry
                    point, if type is LD, or LR
                GBlank if type is ER
            LLength of pseudo-register, if type is PR
                        LBlank-alignment factor for type PR:
                    00=byte, 01=halfword, 03=fullword,
                        07=doubleword alignment
                        Low-order 4 bits indicate section
                        attributes for types SD, PC, or CM
                            X'08' - Read-only csect (RSECT) attribute
                        X'04' - Resident-mode csect attribute
                (0=RMODE 24, 1=RMODE ANY)
                        X'03' - Addressing-mode csect attribute
                        B'00' - AMODE 24
                                B'01' - AMODE 24
                                B'10' - AMODE 31
                                B'11' - AMODE ANY
            L24-bit assigned address of the SD, PC, CM, LD, or LR item
            Type-Hex (00=SD,01=LD,02=ER,03=LR,04=PC,05=CM,06=PR,0A=WX)
    LName--when type is SD,LD,LR,ER,CM,PR
    LBlank}\mp@subsup{}{}{-}\mathrm{ -when type is PC or blank CM.
Mnemonic Codes:
    SD Name of a control section
    LD Name of an entry point within a control section
    ER Name of an external symbol
    LR Name of an entry point within a control section
    PC Private(blank-name)control section
    CM Name of a program common area
    PR Name of a pseudo-register area
    WX Name of an external symbol weakly referenced
```

May 1983

Text Input Record (Card Image)

## CSI Input Record (Card Image)



Note: The record following the CSI record consists entirely of text data, the length of which is given on the CSI record. This record may be up to 32767 bytes long.

RLD Input Record (Card Image)


```
RLD Data Item
```



```
            Flag field--(TTTTLLST)
                TTTT=type S=Direction of relocation
            0000=non-branch(A adcon) 0=positive (+)
            0001=branch(V adcon) 1=negative (-)
            0010=pseudo-register
            0011=cumulative length T=type of next RLD item
                                    0=next RLD item has a
                                    different R or P pointer
            LL=length of address They are present in the
                constant next item.
            01=2 bytes
                1=next RLD item has the
            10=3 bytes same R and P pointers,
            11=4 bytes hence they are omitted.
    Position pointer (P)-ESDID of
        the control section that contains
        the address constant
    LRelocation pointer (R)-ESDID of
        the the symbol being referred to. Zero (00)
        if type=PR cumulative length
```


## END Input Record (Card Image)



SYM Input Record (Card Image)


Symbolic Debugging System Data Item (AG1 and AT1)

| OR | AA | SYM | F | LD | M | S |
| :--- | :--- | :--- | :--- | :--- | :--- | :--- |

Only the OR and AA fields are required for each entry. The presence of the remaining fields is indicated by the OR field.

OR (1 byte) Organization Byte
Bit 0 - If 0, Not data type. Bits 1-3 indicate the following:
Bits 1-3 000 Space (next byte contains the length of the space)
001 Control Section
010 Dummy Control Section
011 Common Section
100 Instruction
101 Channel Command Word
110 Not Used
111 Not Used

If 1, Data type (indicates presence of $F$ and LD fields). Bits 1-3 indicate the following:

Bit 1 If 0, no duplication factor present
If 1, duplication factor present (indicates presence of M field)

Bit 2 If bit $1=0$
0 entry is not a cluster ${ }^{1}$ subfield
1 entry is a cluster subfield
If bit $1=1$
0 entry is not a cluster header
1 entry is a cluster header
Bit 3 If 0, no scale factor present
If 1, scale factor present (indicates presence of $S$ field)

Bit 4 If 0, symbol present (indicates presence of SYM field) If 1, symbol not present

Bits 5-7 Length of symbol (length of SYM field) minus one
AA (3 bytes) Displacement from Base of Control Section
SYM (1-8 bytes) Symbol

May 1983

```
F (1 byte) Format
    0 0 ~ C h a r a c t e r ~ ( C )
    0 4 ~ H e x a d e c i m a l ~ ( X )
    0 8 \text { Binary (B)}
    OC Extended floating-point (L)
    10 Fixed-point (F)
    14 Fixed-point (H)
    18 Floating-point (E)
    1C Floating-point (D)
    20 Address constant (A)
    24 Address constant (Y)
    28 Address constant (S)
    2C Address constant (V)
    30 Packed Decimal (P)
    34 Zoned Decimal (Z)
    3 8 ~ N o t ~ U s e d
    3C Not Used
    4 0 ~ N o t ~ U s e d
    4 4 ~ N o t ~ U s e d
    4 8 \text { Relocatable EQU}
    0 1 ~ B i t ~ l e n g t h ~ f l a g
    0 2 ~ E x p l i c i t ~ l e n g t h ~ f l a g ~
```

            The bit length and explicit length flags are ORed into the
            format field.
    LD (1 or 2 bytes) Data Length (actual length minus one)
If the format of the data is character, hexadecimal, or binary, the
LD field is two bytes.
M (3 bytes) Duplication Factor
S (2 bytes) Scale Factor
${ }^{1}$ A cluster is defined as a multiple constant of variable-length entries
in the format
name $D C \quad m\{C|X| P \mid Z\}^{\prime} n 1, n 2, \ldots \prime$
The cluster header entry specifies the symbol "name", the duplication
factor "m", and the length of the entire data entry. The cluster
subfield entries specify the data format (either $C, X, P$, or $Z$ ) and the
length of each element in the data entry.

Symbolic Debugging System Data Item (AG2)

| OR | AA | SYM | F | LD | M | S |
| :--- | :--- | :--- | :--- | :--- | :--- | :--- |

Only the OR and AA fields are required for each entry. The presence of the remaining fields is indicated by the OR field.

OR (1 byte) Organization Byte
Bit 0 - If 0, Not data type. Bits 1-3 indicate the following:
Bits 1-3 000 Space (next byte contains the length of the space)
001 Control Section
010 Dummy Control Section
011 Common Section
100 Instruction
101 Channel Command Word
110 EQU, LTORG, CNOP, ORG (indicates presence of $F$ and LD fields)
111 Not Used

- If 1, Data type (indicates presence of $F$ and LD fields). Bits 1-3 indicate the following:

Bit 1 If 0, no duplication factor present
If 1, duplication factor present (indicates presence of $M$ field)

Bit 2 If bit $1=0$
0 entry is not a cluster ${ }^{1}$ subfield
1 entry is a cluster subfield
If bit $1=1$
0 entry is not a cluster header
1 entry is a cluster header
Bit 3 If 0, no scale factor present
If 1, scale factor present (indicates presence of S field)

Bit 4 If 0, symbol present (indicates presence of SYM field) If 1, symbol not present

Bits 5-7 Length of symbol (length of SYM field) minus one
AA (3 bytes) Displacement from Base of Control Section SYM (1-8 bytes) Symbol

```
F (1 byte) Format
    0 0 ~ C h a r a c t e r ~ ( C ) ~
    0 4 ~ H e x a d e c i m a l ~ ( X )
    0 8 ~ B i n a r y ~ ( B )
    0C Not Used
    10 Fixed-point (F)
    14 Fixed-point (H)
    18 Floating-point (E)
    1C Floating-point (D)
    20 Address constant (A)
    24 Address constant (Y)
    28 Address constant (S)
    2C Address constant (V)
    30 Packed Decimal (P)
    34 Zoned Decimal (Z)
    38 Extended floating-point (L)
    3C Not Used
    4 0 ~ N o t ~ U s e d
    4 4 ~ N o t ~ U s e d
    4 8 \text { Not Used}
    0 1 ~ B i t ~ l e n g t h ~ f l a g
    0 2 ~ E x p l i c i t ~ l e n g t h ~ f l a g ~
    xx User-supplied EQU value if bits 0-3 of OR are 0110
        (xx is OO if user-supplied value is not given)
        The bit length and explicit length flags are ORed into the
        format field.
LD (1 or 2 bytes) Data Length (actual length minus one)
    If the format of the data is character, hexadecimal, binary, or
    if bits 0-3 of OR are 0110, the LD field is two bytes. If the
    data length is bit length, the length is given in bits; otherwise,
    it is given in bytes.
M (3 bytes) Duplication Factor
S (2 bytes) Scale Factor
```

${ }^{1}$ A cluster is defined as a multiple constant of variable-length entries
in the format
name $D C \quad m\{P \mid Z\}^{\prime} n 1, n 2, \ldots \prime$
The cluster header entry specifies the symbol "name", the duplication
factor "m", and the length of the entire data entry. The cluster
subfield entries specify the data format (either $P$ or $Z$ ) and the length
of each element in the data entry.

Symbolic Debugging System Data Item (AH1)

| OR | AA | LN | SYM | F | LD | M | S |
| :--- | :--- | :--- | :--- | :--- | :--- | :--- | :--- |

Only the OR and AA fields are required for each entry. The presence of the remaining fields is indicated by the OR field.

OR (1 byte) Organization Byte
Bit 0 - If 0, Not data type. Bits 1-3 indicate the following:
Bits 1-3 000 Space (next byte contains the length of the space)
001 Control Section
010 Dummy Control Section
011 Common Section
100 Instruction
101 Channel Command Word
110 EQU, LTORG, CNOP, ORG (indicates presence of $F$ and LD fields)
111 Not Used

- If 1, Data type (indicates presence of $F$ and LD fields). Bits 1-3 indicate the following:

Bit 1 If 0, no duplication factor present
If 1, duplication factor present (indicates presence of M field)

Bit 2 If bit $1=0$
0 entry is not a cluster ${ }^{1}$ subfield
1 entry is a cluster subfield
If bit $1=1$
0 entry is not a cluster header
1 entry is a cluster header
Bit 3 If 0 , no scale factor present
If 1, scale factor present (indicates presence of S field)

Bit 4 If 0, symbol present (indicates presence of LN and SYM fields)
If 1, symbol not present
Bits 5-7 Bit offset
AA (3 bytes) Displacement from Base of Control Section
LN (1 byte) Length of Symbol (actual length minus one)

May 1983

```
SYM (1-63 bytes) Symbol
F (1 byte) Format
0 0 ~ C h a r a c t e r ~ ( C ) ~
0 4 ~ H e x a d e c i m a l ~ ( X )
0 8 ~ B i n a r y ~ ( B )
0C Not Used
10 Fixed-point (F)
14 Fixed-point (H)
18 Floating-point (E)
1C Floating-point (D)
20 Address constant (A)
24 Address constant (Y)
28 Address constant (S)
2C Address constant (V)
30 Packed Decimal (P)
34 Zoned Decimal (Z)
38 Extended floating-point (L)
3C Not Used
4 0 ~ N o t ~ U s e d ~
4 4 ~ N o t ~ U s e d
4 8 \text { Not Used}
0 1 ~ B i t ~ l e n g t h ~ f l a g
02 Explicit length flag
xx User-supplied EQU value if bits 0-3 of OR are 0110
        (xx is 00 if user-supplied value is not given)
                The bit length and explicit length flags are ORed into the
                format field.
LD (1 or 2 bytes) Data Length (actual length minus one)
    If the format of the data is character, hexadecimal, binary, or
    if bits 0-3 of OR are 0110, the LD field is two bytes. If the
    data length is bit length, the length is given in bits; otherwise,
    it is given in bytes.
M (3 bytes) Duplication Factor
S (2 bytes) Scale Factor
```

${ }^{1}$ A cluster is defined as a multiple constant of variable-length entries
in the format
name $D C \quad m\{P \mid Z\}^{\prime} n 1, n 2, \ldots \prime$
The cluster header entry specifies the symbol "name", the duplication
factor "m", and the length of the entire data entry. The cluster
subfield entries specify the data format (either $P$ or $Z$ ) and the length
of each element in the data entry.

Symbolic Debugging System Data Item (PL3)

| OR | AA | SYM | F | LD | M |
| :--- | :--- | :--- | :--- | :--- | :--- |

Only the OR and AA fields are required for each entry. The presence of the remaining fields is indicated by the OR field.

OR (1 byte) Organization Byte
Bit 0 - If 0, Not data type. Bits 1-3 indicate the following: Bits 1-3 000 Not Used

001 Control Section
010 Dummy Control Section
011 Common Section
100 Instruction
101 Not Used
110 Not Used
111 Not Used

- If 1, Data type (indicates presence of $F$ and LD fields). Bits 1-3 indicate the following:

Bit 1 If 0, no duplication factor present
If 1, duplication factor present (indicates presence of M field)

Bit 2 Not Used Bit 3 Not Used

Bits 4-7 Length of symbol (length of SYM field)
AA (3 bytes) Displacement from Base of Control Section
SYM (1-15 bytes) Symbol
F (1 byte) Format
00 Not Used
04 Hexadecimal
08 Not Used
0C Not Used
10 Fixed-point (Fullword)
14 Fixed-point (Halfword)
18 Floating-point (Short)
1C Floating-point (Long)

May 1983

LD (1 or 2 bytes) Data Length (actual length minus one)
If the format of the data is hexadecimal, the LD field is two bytes.

M (3 bytes) Duplication Factor

Symbolic Debugging System Data Item (FG1)


Only the OR and AA fields are required for each entry. The presence of the remaining fields is indicated by the OR field.

OR (1 byte) Organization Byte
Bit 0 - If 0, Not data type. Bits 1-3 indicate the following:
Bits 1-3 000 Not Used
001 Control Section
AA field contains the length of the TEMP \& CONSTANT pool in the FORTRAN object module (required for object time dimensions)
010 Not Used
011 Common Section
100 Not Used
101 Not Used
110 Internal Statement Number (source listing)
111 External Statement Number (statement label)

- If 1, Data type (indicates presence of $F$ and LD fields). Bits 1-3 indicate the following:

Bit 1 Not Used
Bit 2 If 0 , entry is not an array
If 1, entry is an array (indicates presence of $A R$ and MUL fields)

Bit 3 If 0, entry is not a call by name dummy argument
If 1, entry is a call by name dummy argument in a FORTRAN subprogram. AA field gives the displacement to a pointer to the argument location in the FORTRAN calling program.

Bit 4 If 0, symbol present (indicates presence of SYM field) If 1 , symbol not present

May 1983

Bits 5-7 Length of symbol (length of SYM field) minus one
AA (3 bytes) Displacement from Base of Control Section
SYM (1-6 bytes) Symbol
F (1 byte) Format
04 Hexadecimal
10 Integer (Fullword)
14 Integer (Halfword)
18 Real (Short)
1C Real (Long)
38 Complex (Short)
3C Complex (Long)
40 Logical (Fullword)
44 Logical (Byte)
LD (1 byte) Data Length (actual length minus one)
If the format of the data is hexadecimal, the LD field is two bytes.

AR (1 byte) Dimension Size (number of subscripts)
The dimension size indicates number of MUL fields (1 - 7)

MUL (4 bytes) Multiplier for Array Subscript
Bit 0 - If 0, MUL field (bits 1-31) contains the actual multiplier used to calculate array subscripts
If 1, MUL field (bits 1-31) contains the displacement to a pointer in the TEMP \& CONSTANT pool in the FORTRAN oject module. This is used for object time dimensions.

Symbolic Debugging System Data Item (PL1)

| OR | LI | INFO | LN | NAME |
| :--- | :--- | :--- | :--- | :--- |

There are 13 types of data items for PL1 SYM records. The general format is given above. Only the OR and LI fields are required; the INFO, LN, and NAME fields are only present for certain types.

OR (1 byte) Organization byte
00 Control section
01 Pseudo-register
03 Statement (labeled)
05 Entry or procedure label
x7 Label variable
08 File
xF Data variable
80 Procedure or ON-unit
81 Begin statement
82 Entry statement
83 Statement (unlabeled)
84 Iterative DO statement
85 End statement
LI (1 byte) Length of INFO field
INFO (n bytes) Information field (present if LIFO), n=LI bytes
LN (1 byte) Length of NAME field minus one
NAME ( n bytes) Symbolic name, $\mathrm{n}=\mathrm{LN}+1$ bytes
The format for each type is given below:
00 Control section


OR (1 byte) Organization byte (hex 00)
LI (1 byte) Length of INFO (hex 00)
LN (1 byte) Length of NAME minus one
NAME ( n bytes) Control section name, $\mathrm{n}=\mathrm{LN}+1$
01 Pseudo-register

| OR | LI | RCNT | LN | NAME |
| :--- | :--- | :--- | :--- | :--- |


| OR | (1 byte) | Organization (hex 01) |
| :--- | :--- | :--- |
| LI | $(1$ byte) | Length of INFO (hex 02) |
| RCNT | $(2$ bytes $)$ | Reference count |

LN (1 byte) Length of NAME minus one
NAME (n bytes) Pseudo-register name, $n=L N+1$
03 Statement (labeled)

| OR | LI | BC | STA\# | ADDR | LN | NAME |
| :--- | :--- | :--- | :--- | :--- | :--- | :--- |


| OR | (1 byte) | Organization byte (hex 03) |
| :--- | :--- | :--- | :--- |
| LI | (1 byte) | Length of INFO (hex 06) |
| BC | (1 byte) | Block count |
| STA\# | $(2$ bytes) | Statement number |
| ADDR | $(3$ bytes) | Displacement in program control section |
| LN | $(1$ byte) | Length of statement label minus one |
| NAME | $(n$ bytes) | Statement label name, $n=L N+1$ |

05 Entry or procedure label

| OR | LI | BC | STA\# | ADDR | DED | LN |
| :--- | :--- | :--- | :--- | :--- | :--- | :--- |

NAME

OR (1 byte) Organization byte (hex 05)
LI (1 byte) Length of INFO (hex 09)
BC (1 byte) Block count
STA\# (2 bytes) statement number
ADDR (3 bytes) Displacement in program control section
DED (3 bytes) Data element descriptor
LN (1 byte) Length of entry or procedure label minus one NAME (n bytes) Entry or procedure label name, $n=L N+1$
x7 Label variable

| OR | LI | BC | RCNT | STA\# | ADDR | DF |
| :--- | :--- | :--- | :--- | :--- | :--- | :--- |


| DM | DVA | LN | NAME | DTAB |
| :--- | :--- | :--- | :--- | :--- |


| OR | $(1$ byte) | Organization byte (hex x7) <br> data variable entry) |
| :--- | :--- | :--- |
| LI see OR field for |  |  |


| DF | $(1$ byte) | Data flag (see DF field of data variable entry) |
| :--- | :--- | :--- |
| DM | $(1$ byte) | Dimensionality |
| DVA | $(3$ bytes) |  |

08 File

| OR | LI | RCNT | STA\# | ADDR | LN |
| :--- | :--- | :--- | :--- | :--- | :--- |

## NAME

OR (1 byte) Organization byte (hex 08)
LI (1 byte) Length of INFO (hex 07)
RCNT (2 bytes) Reference count
STA\# (2 bytes) Statement number
ADDR (3 bytes) Displacement of DCLCB in program control
section. If zero, file is external and refers
to last control section.
LN (1 byte) Length of NAME minus one
NAME (n bytes) File name, $n=L N+1$
xF Data variable

| OR | LI | BC | RCNT | STA\# | ADDR | DF |
| :--- | :--- | :--- | :--- | :--- | :--- | :--- |


| DM | DED | NUM | DEDA | DVA |
| :--- | :--- | :--- | :--- | :--- |


| LN | NAME | LB | BPTR | DTAB |
| :--- | :--- | :--- | :--- | :--- |

OR (1 byte) Organization byte (hex xF)
bit 0: 1, if no symbolic name
1: Unused
2: Unused
3: 1, if dimensioned
LI (1 byte) Length of INFO
BC (1 byte) Block count


80 Procedure or on-unit

| OR | LI | BL | BC | CB | STA\# | ADDR | $\mathrm{P} \#$ |
| :--- | :--- | :--- | :--- | :--- | :--- | :--- | :--- |


| LEN | OP | LN1 | NAM1 |
| :--- | :--- | :--- | :--- |

OR (1 byte) Organization byte (hex 80)

LI (1 byte) Length of INFO (hex 0D)
BL (1 byte) Block level
BC (1 byte) Block count
CB (1 byte) Containing block count
STA\# (2 bytes) statement number
ADDR (3 bytes) Displacement in program control section
P\# (1 byte) Number of symbolic parameters
LEN (3 bytes) Length of procedure
OP (1 byte) Procedure options

$$
\begin{array}{lll}
\text { bit } 0: & 1, & \text { if REENTRANT } \\
1: & 1, & \text { if ON-UNIT } \\
2: & 1, & \text { if MAIN } \\
3: & 1, & \text { if TASK } \\
4: & 1, & \text { if RECURSIVE } \\
5: & 1, & \text { if OPTIONS } \\
6: & 1, & \text { if statement RETURN expression } \\
7: & 1, & \text { if an argument is ENTRY }
\end{array}
$$

LN1 (1 byte) Length of parameter symbol 1 minus one NAM1 (n bytes) Parameter symbol 1, n=LN1+1
LN2 (1 byte) Length of parameter symbol 2 minus one NAM2 (n bytes) Parameter symbol 2, n=LN2+1
-••••
.. ...
81 Begin statement

| OR | LI | BL | BC | CB | STA\# | ADDR |
| :--- | :--- | :--- | :--- | :--- | :--- | :--- |


| OR | (1 byte) | Organization byte (hex 81) |
| :--- | :--- | :--- | :--- |
| LI | (1 byte) | Length of INFO (hex 08) |
| BL | (1 byte) | Block level |
| BC | (1 byte) | Block count |
| CB | (1 byte) | Containing block count |
| STA\#\# | $(2$ bytes) | Statement number |
| ADDR | $(3$ bytes) | Displacement in program control section |

82 Entry statement

| OR | LI | BL | BC | CB | STA\# | ADDR | P\# |
| :--- | :--- | :--- | :--- | :--- | :--- | :--- | :--- |



OR (1 byte) Organization byte (hex 82)
LI (1 byte) Length of INFO (hex 09)
BL (1 byte) Block level
BC (1 byte) Block count
CB (1 byte) Containing block count
STA\# (2 bytes) statement number
ADDR (3 bytes) Displacement in program control section
P\# (1 byte) Number of symbolic parameters
LN1 (1 byte) Length of parameter symbol 1 minus one
NAM1 (n bytes) Parameter symbol 1, n=LN1+1
LN2 (1 byte) Length of parameter symbol 2 minus one
NAM2 (n bytes) Parameter symbol 2, n=LN2+1
. . . . .
. . . . .

83 Statement (unlabeled)

| OR | LI | BC | STA\# | ADDR |
| :--- | :--- | :--- | :--- | :--- |

OR (1 byte) Organization byte (hex 83)
LI (1 byte) Length of INFO (hex 06)
BC (1 byte) Block count
STA\# (2 bytes) statement number
ADDR (3 bytes) Displacement in program control section
84 Iterative DO statement

| OR | LI | BC | STA\# | ADDR |
| :--- | :--- | :--- | :--- | :--- |


| OR | (1 byte) | Organization byte (hex 84) |
| :--- | :--- | :--- |
| LI | $(1$ byte) | Length of INFO (hex 06) |
| BC | $(1$ byte) | Block count |
| STA\# | $(2$ bytes) | Statement number |
| ADDR | $(3$ bytes) | Displacement in program control section |

MTS 5: System Services

85 End statement

```
\begin{tabular}{|l|l|l|l|l|}
\hline OR & LI & EC & STA\# & ADDR \\
\hline
\end{tabular}
    OR (1 byte) Organization byte (hex 85)
LI (1 byte) Length of INFO (hex 06)
EC (1 byte) End code
00 - end of procedure
01 - end of BEGIN block
02 - end of iterative DO section
STA# (2 bytes) Statement number
ADDR (3 bytes) Displacement in program control section
Note: The formats of the declare control block (DCLCB) and the data
element descriptor (DED) are given in IBM System/360 Operating
System PL/I Subroutine Library Program Logic Manual, form GY28-6801.
```

User-Generated Load Records

LDT Input Record (Card Image)


REP Input Record (Card Image)


Example: REP 001A54 01,1A23,47FDA123 Patch to AR 2,3; B LOOP

DEF Input Record (Card Image)


ENT Input Record (Card Image)


May 1983

NCA Input Record (Card Image)


COM Input Record (Card Image)

| 1 | 2-4 | 5-80 |
| :---: | :---: | :---: |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

MDL Input Record (Card Image)


LCS Input Record (Card Image)


Note that the system low-core symbol dictionary has the predefined name LCSYMBOL. An LCS card referring to that symbol can be placed at the end of the deck by the user and will reduce the loading time if no subroutines are required from *LIBRARY. Such a record is available in the public file *LCS.

May 1983

MSG Input Record (Card Image)

| 1 | 2-4 | 5 | 6-80 |
| :---: | :---: | :---: | :---: |
|  |  |  |  |
| \| | \| | \| | -M |
| \| | \| | \| |  |
| \| | \| |  |  |
| \| |  |  | ot us |
|  |  |  |  |
| \| | -M |  |  |
|  | t us |  |  |

## OPT Input Record (Card Image)

| 1 | 2-4 | 5 | 6-80 |
| :---: | :---: | :---: | :---: |
|  |  |  |  |
| \| |  | \| | டLi |
| 1 | \| | 1 |  |
| \| | \| |  |  |
| \| |  |  | ot use |
| 1 | -OP |  |  |
|  | t us |  |  |

Note: See the subsection "Modifying Programs in Object Module Form" for a description of the loader options available.

ALI Input Record (Card Image)


Library Control Records

LIB Input Record (Card Image)

| 1 | 2-4 | 5-8 | 9-15 | 16 | 17-24 | 25-40 | 41-80 |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  |  |  |  |  |  |  |
|  |  |  | \| | \| |  |  |  |
| \| | \| | \| | \| | \| |  |  | —Not |
| \| | \| | \| | \| | \| |  |  |  |
| \| | \| |  |  |  |  | $\llcorner$ (opti | nal) not |
| \| | \| |  | \| |  |  | first | record |
| \| |  |  |  |  |  | objec | module |
|  | \| | \| | \| | \| |  |  |  |
| \| |  |  | \| | \| | LThe | odule n | ne of |
| \| |  |  |  |  | the | bject m | dule. |
|  |  |  | \| |  |  |  |  |
| \| |  |  |  |  | if not | -point | nformat |
| \| |  |  |  |  | lank if | informa | ion not |
|  |  |  |  |  |  |  |  |
| I |  |  | —Not | us |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  | $\square \mathrm{B}$ | lank: | obj | ct modu | e foll |  |
| \| |  | ᄂFu | ullwor | d in | teger n | mber co | isting |
| \| |  |  | he fir | st | ine of | he obj | modul |
|  |  |  |  |  |  |  |  |
| \| | -L |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  | -Not | used |  |  |  |  |  |

## RIP Input Record (Card Image)

| 1 | \| - 4 | 5-6 | 7-14 | 15-16 | 17-24 | 25-80 |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| 1 | + |  |  |  |  |  |
| \| | \| | \| | \| | \| |  | —Not |
| \| | \| | \| | \| | \| |  |  |
| \| | \| |  |  |  | LExt | nal sy |
| \| | \| |  | \| |  |  |  |
| \| | \| | \| | \| | -Not | used |  |
| \| | \| | \| |  |  |  |  |
| \| | \| | \| | -Ex | ternal | symbol | to be |
| \| | \| | , | to | , but | not def | ned, i |
| \| | \| | \| | to | be tes | sted is | of tha |
|  | \| | \| | bl | ank. |  |  |
|  | \| |  |  |  |  |  |
|  | \| | -No | ot us |  |  |  |
| 1 | $\llcorner\mathrm{R}$ |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  | Not | used |  |  |  |  |

DIR Input Record (Card Image)


The actual directory is stored as the next record and consists of entries, each with the following format:

Directory Entry


Note: The directory entry record may be up to 32767 bytes long.

MTS 5: System Services

May 1983

VIRTUAL MEMORY MANAGEMENT

The amount of virtual memory allocatable for each user is currently 15 segments. Each segment consists of 256 pages of 4096 bytes per page. Currently, virtual memory has the following layout:

| Segment | $0:$ | reserved for nonrelocatable nonpaged resident system <br> routines. |
| :--- | ---: | :--- |
| Segments $1-5:$ | reserved for sharable paged resident system routines. |  |
| Segment | $6:$ | unused. |
| Segment | $7:$ | system allocated storage for user's task. |
| Segments $8-14:$ | available for allocation by user's task. |  |

Since the charge for virtual memory is a function of both size and time of use, the user may wish to economize on the use of virtual memory. Instead of initially allocating all the virtual memory needed for a program at the time the program is loaded and retaining it until termination, memory can be obtained and released dynamically (during execution). This section discusses the various ways of accomplishing this.

Calling the system subroutine GETSPACE is the primary method for dynamically allocating virtual memory. This subroutine is called implicitly by the \$RUN, \$RERUN, \$DEBUG, and \$LOAD commands. These commands call on the loader to load one or more object modules; the loader in turn calls on GETSPACE to acquire space for the object modules being loaded. Object modules consist of control sections each of which must occupy a contiguous block of virtual memory. The number of control sections in an object module depends on the compiler used. If there is more than one control section in a module, the loader only insists that each control section get a contiguous section of memory. The loader will make a separate call to GETSPACE to acquire memory space for each control section. Therefore, each control section may be arbitrarily located in virtual memory. Although the module is logically related, its control sections may be physically dispersed throughout virtual memory. Virtual memory acquired for programs loaded by the \$RUN, \$RERUN, \$DEBUG, and \$LOAD commands is managed by the system.

The loader (and therefore GETSPACE) is also called implicitly by the system subroutines LINK, LOAD, and XCTL. The subroutine LINK can be called to load a program into memory, execute that program, and after execution, release the acquired memory and return to the calling program. The memory acquired by the LINK subroutine is managed by the system. The subroutine LOAD can be called to load an object module into that memory without transferring to the entry point. The user has the option of managing this memory himself or allowing the system to manage it for him. The subroutine XCTL can be called to release the memory of the program that called XCTL, load a new program into memory, and
transfer to the entry point of the new program. This memory is managed by the system.

The subroutine GETSPACE can also be called explicitly by a user's program. Virtual memory acquired explicitly by the user can be managed by the user or by the system, at the user's option. With a call to GETSPACE, the user specifies the size of memory desired, the segment from which the memory is to be allocated, and the manner in which the memory is to be managed. These concepts are explained below.

There are basically four ways to release virtual memory:
(1) by the termination of execution of the entire program,
(2) implicitly, by returning from a program loaded via the subroutines LINK or XCTL,
(3) implicitly, by calling the subroutine XCTL, or
(4) explicitly, by calling the subroutines FREESPAC or UNLOAD or by using the command \$UNLOAD.

Since control sections may be widely dispersed in virtual memory, they must be logically connected. The concept of storage index numbers and link levels serves the purpose of identifying logically connected control sections. Each area of virtual memory allocated has an associated storage index number, ranging from 00 to FF (base 16). All logically connected areas (those control sections loaded with one call to the loader) are given the same storage index number. Later, when memory is released, these numbers serve as identifiers for determining which sections of memory are released. The maximum storage index number is the largest storage index number assigned by the loader to a block of memory (that has not yet been released). When a new program is loaded, this storage index number is incremented by one to form a new maximum storage index number.

A link level is a group of storage index numbers associated with the modules loaded between successive calls on the subroutines LINK or XCTL. When a new link level is started, the number assigned as the link level index is the storage index number of the block of memory being loaded (the maximum storage index index). This number serves as a base for referring to all storage index numbers that belong to that link level. All storage index numbers assigned to new blocks of memory loaded until a new link level is started belong to this link level. A pushdown table is maintained that records all the storage index numbers associated with each link level. A call to the subroutine LINK starts a new link level; a return from LINK reactivates a previous link level. The current link level index is the current level in the table (the link level of the modules currently executing). The top of the current link level is the maximum storage index number that has been currently assigned.

Virtual memory is logically divided into two sections. The highest storage level section is memory that is available for the entire program until termination or until explicitly released. This section of memory is designed for reference by programs executing at any link level. The current storage level section is memory that is available for reference
by a part of the program executing at the corresponding link level. Blocks of current storage level memory are released when the link level that allocated them returns to a previous level. The storage index numbers play a role in the management of these two sections of memory. The numbers in the range 00 to 7 F (base 16) are the highest storage level numbers. They are assigned only to memory acquired through the GETSPACE and LOAD subroutines when the option for "highest" is chosen. These areas are released only when explicitly requested or when execution of the entire program terminates. The storage index numbers in the range 80 to $F F$ (base 16) are the current storage level numbers. These are assigned via the subroutines GETSPACE and LOAD (with the "current" option chosen), LINK, and XCTL. These areas are released automatically under conditions discussed below.

The subroutine LINK can be called explicitly by the user or implicitly by the \$RUN, \$RERUN, \$DEBUG, and \$LOAD commands. When the user calls LINK, the loader is called to load the modules specified and then transfer to the entry point for execution. When the \$RUN, \$RERUN, \$DEBUG, and \$LOAD commands call LINK, the loader is called to load the modules specified, but instead of transferring to the entry point immediately, control is returned to the command for further processing of command options. Execution is started with a special transfer to the entry point for the \$RUN, \$RERUN, and \$DEBUG commands or via the \$START or \$RESTART commands when the \$LOAD command is used. When LINK is called implicitly via the \$RUN, \$RERUN, \$DEBUG, or \$LOAD commands, the initial maximum storage index number is 80 and the current link level index is 80.

When the subroutine LINK is called by the user's program, the name of a file or device containing the object module(s) to be loaded and linked to is given. LINK calls the loader which in turn calls GETSPACE for each control section in the module(s), loads the module(s), and transfers control to the entry point. The maximum storage index number, in the range $80-\mathrm{FF}$, is incremented by one and the allocated memory is assigned this number. This number is now the maximum storage index number, as well as the current link level index, marking the start of the range of storage index numbers assigned to that link. Any calls to the LOAD subroutine which choose the option for the current storage level (range $80-\mathrm{FF}$ ) and occur before the subroutine LINK returns are assigned their own storage index numbers but are considered to be part of the current link level. When the linked module terminates execution, the subroutine LINK returns to the calling program and all memory identified with the current link level is released. Thus, those modules loaded via a call to the LOAD subroutine with the "current" option are released along with the linked routine. The maximum storage index number is then reset to what it was before LINK was called, i.e., to the maximum storage index number associated with the previous link level index. The current link level index is reset to the storage index number assigned to the most recent link that has not yet returned. When the linked modules return after execution to the LINK subroutine, all memory associated with the modules is released by the LINK subroutine and control is returned to the program which called LINK. See the example given by Tables $1(\mathrm{a})$ and $1(\mathrm{~b})$ and Figure 1 below.

Table 1(a)

| Storage <br> Index <br> Number$\|$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| :--- | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| 80 | $M$ | M | M | M | M | M | M | M | M |
| 81 |  | $A$ |  | $B$ | $B$ | $B$ | $B$ | $B$ |  |
| 82 |  |  |  |  | $C$ |  | $D$ |  |  |

Table $1(\mathrm{~b})$

|  | Maximum <br> Storage <br> Index <br> Number | Current <br> Link <br> Level <br> Index |
| :---: | :---: | :---: |
| 0 | 80 | 80 |
| 1 | 81 | 81 |
| 2 | 80 | 80 |
| 3 | 81 | 81 |
| 4 | 82 | 82 |
| 5 | 81 | 81 |
| 6 | 82 | 82 |
| 7 | 81 | 81 |
| 8 |  | 80 |

May 1983

Figure 1

(1) A is loaded and its execution begins.
(2) A is released and control returns to MAIN.
(3) $B$ is loaded and its execution begins.
(4) C is loaded and its execution begins.
(5) C is released and control returns to $B$.
(6) D is loaded and its execution begins.
(7) D is released and control returns to B.
(8) B is released and control returns to MAIN.

The subroutine XCTL performs the same functions as LINK, except that it first releases all storage associated with the current link level (usually the program calling XCTL) before allocating memory and loading the specified module. Instead of incrementing the maximum storage index number and current link level index, these numbers are reassigned to the new modules loaded since the previous modules associated with them have been released. See the example given by Tables 2(a) and 2(b) and Figure 2 below.

Table 2(a)

| Storage <br> Index <br> Number | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |  |
| :--- | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| 80 | $M$ | M | M | M | M | M | M | M | M |  |
| 81 |  | A | B | B | B | B | B | B |  |  |
| 82 |  |  |  |  | C | D |  | E |  |  |

Table $2(\mathrm{~b})$

|  | Maximum <br> Storage <br> Index <br> Number | Current <br> Link <br> Level <br> Index |
| :---: | :---: | :---: |
| 0 | 80 | 80 |
| 1 | 81 | 81 |
| 2 | 81 | 81 |
| 3 | 82 | 82 |
| 4 | 82 | 82 |
| 5 | 81 | 81 |
| 6 | 82 | 82 |
| 7 | 81 | 81 |
| 8 |  | 80 |

May 1983

Figure 2
MAIN

(1) A is loaded and its execution begins.
(2) A is released, B is loaded and its execution begins.
(3) C is loaded and its execution begins.
(4) C is released, D is loaded and its execution begins.
(5) D is released and control returns to B.
(6) E is loaded and its execution begins.
(7) E is released and control returns to B.
(8) B is released and control returns to MAIN.

The subroutine LOAD also acquires space for each control section and loads the specified object module(s). However, it does not transfer control to the module it loads. Instead, it returns the entry point of the module so that it can be called as a subroutine or referred to by another program. By selecting the option "current", the call on LOAD can request that memory allocated be considered as part of the current link level. This means that when the program returns from the current link level, the memory acquired by LOAD is also released. The storage index numbers assigned to the blocks of memory allocated by LOAD differ from the other storage index numbers of the current link level so that these blocks of memory may be released separately by the UNLOAD subroutine. However, these loaded modules are nested within the current link level and their memory is released when that link level is released, if not previously released by UNLOAD. (LOAD may also be called with the "highest" option in which case storage is not automatically released. This option is discussed below.) See the example given in Tables $3(\mathrm{a})$ and $3(\mathrm{~b})$ and Figure 3 below.

Table 3(a)

| Storage <br> Index <br> Number | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |  |
| :--- | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| 80 | $M$ | M | M | M | M | M | M | M | M | M |  |
| 81 |  | A | A | A | A | A | A | A | A |  |  |
| 82 |  |  |  | B | B | B | B | B | B | B |  |
| 83 |  |  |  | C | C | C | C | C | C |  |  |
| 84 |  |  |  |  | D | D |  | F |  |  |  |
| 85 |  |  |  |  |  | E |  |  |  |  |  |

Table 3 (b)

|  | Maximum <br> Storage <br> Index <br> Number | Current <br> Link <br> Level <br> Index |
| :---: | :---: | :---: |
| 0 | 80 | 80 |
| 1 | 81 | 81 |
| 2 | 82 | 81 |
| 3 | 83 | 81 |
| 4 | 84 | 84 |
| 5 | 85 | 84 |
| 6 | 83 | 81 |
| 7 | 84 | 84 |
| 8 | 80 | 81 |
| 9 |  | 80 |

May 1983

Figure 3

(1) A is loaded and its execution begins.
(2) $B$ is loaded.
(3) C is loaded.
(4) D is loaded and its execution begins.
(5) E is loaded.
(6) D and E are released and control returns to A.
(7) $F$ is loaded and its execution begins.
(8) $F$ is released and control returns to A.
(9) A, B, and C are released and control returns to MAIN.

The example given in Tables $4(\mathrm{a})$ and $4(\mathrm{~b})$ and Figure 4 below illustrate the subroutines LINK, LOAD, and XCTL used together.

Table 4(a)

| Storage <br> Index <br> Number | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |  |
| :--- | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| 80 | $M$ | M | M | M | M | M | M | M | M |  |
| 81 |  | A | A | C | C | C | C | F |  |  |
| 82 |  |  |  | $B$ |  | D | D |  |  |  |
| 83 |  |  |  |  |  | E |  |  |  |  |

Table 4(b)

|  | Maximum <br> Storage <br> Index <br> Number | Current <br> Link <br> Level <br> Index |
| :---: | :---: | :---: |
| Time | 80 | 80 |
| 1 | 81 | 81 |
| 2 | 82 | 81 |
| 3 | 81 | 81 |
| 4 | 82 | 82 |
| 5 | 83 | 82 |
| 6 | 81 | 81 |
| 7 | 81 | 81 |
| 8 |  | 80 |

May 1983

Figure 4
MAIN

(1) A is loaded and its execution begins.
(2) $B$ is loaded.
(3) A and B are released, $C$ is loaded and its execution begins.
(4) D is loaded and its execution begins.
(5) E is loaded.
(6) $D$ and E are released and control returns to $C$.
(7) C is released, $F$ is loaded and its execution begins.
(8) F is released and control returns to MAIN.

If the user wishes to load a module via the subroutine LOAD and not to have it released when the memory for the current link level is released, he can call LOAD with the "highest" option. With this option, the memory acquired is assigned the smallest available storage index number in the range 01 to 7 F (base 16). Areas with storage numbers in this range are not released until the program execution terminates normally or the user explicitly requests the release via the subroutines FREESPAC or UNLOAD or the command \$UNLOAD.

As mentioned above, GETSPACE can be called explicitly by the user. In the call to GETSPACE, the user can specify whether the memory should be associated the current storage level or the highest storage level. If the highest storage level is specified, storage index number 00 is always used; if the current storage level is chosen, the storage index number used by the last call to LINK or XCTL is used. As expected, this choice affects when the memory is to be released. If the current storage level is chosen, the memory is automatically released at the next return from a LINK or the next call to XCTL. Selecting the highest storage level means that the memory is not released until execution terminates or an explicit request is made. GETSPACE returns the address of the first byte of the memory region acquired to the calling program. The call to GETSPACE has additional options for the user. The user may specify in which segment space is to be allocated. Ordinarily, space is merely allocated wherever it is available. The user may also assign the storage index number.

Thus far, the discussion has concentrated on how to acquire space and how it is released automatically. There are also three ways to request the release of memory explicitly.

The subroutine FREESPAC is used in conjunction with GETSPACE. A call to FREESPAC releases all or part of a contiguous block of virtual memory that was acquired via GETSPACE. The user gives the address of the first byte of the region to be released and the length to be released. If the length is omitted, the entire region is released.

The subroutine UNLOAD releases memory that was acquired through a previous call to the LOAD subroutine. The region to be released may be specified by file name, external symbol, virtual memory address, or storage index number. When the memory is released, the storage index number assigned to it is also released. The FREESPAC subroutine should not be used to release memory obtained by the LINK, LOAD, and XCTL subroutines. If the storage index number is in the highest range (00-7F), it will be reused on the next call to the LOAD subroutine which requests the "highest" option. However, if the number is in the "current" range, this is not necessarily so. A call to LOAD at "current level" uses the smallest unused storage index number that is greater than or equal to the current link level index. If the number released belongs to a different link level, it is not reused immediately.

Finally, the MTS command \$UNLOAD releases all virtual memory allocated by the user's program.

THE LINKAGE EDITOR

The linkage editor in MTS provides a wide range of facilities for manipulating programs which are in object module form. These facilities fall roughly into three categories: making physical changes to object programs, making logical changes to object programs, and examining the structure of object programs.

Physical changes to an object program have no effect on its execution. A large number of physical formats can represent the same object program. The linkage editor provides facilities for changing from one physical format to another so that:
(1) an object program will fit on an alternative storage medium (e.g., from a line file to the card punch);
(2) the CPU time required to load an object program is minimized;
(3) the file space required to store an object program is minimized.

The linkage editor also provides facilities for changing the logical structure of object programs. Symbols in object programs can be replaced or deleted. Object modules can be combined, deleted, or replaced. The loader-defined entry point can be changed.

Interrogative facilities are also available for examining the logical structure of object programs. Maps and cross-reference tables of the symbols defined and referenced in an object program can be displayed. In addition, object module text may be displayed, scanned, or modified.

The linkage editor is available in the file *LINKEDIT and is invoked via the \$RUN command. The linkage editor uses the following MTS logical I/O units:

```
SCARDS - either the input file containing the object modules to be
    edited, or a sequence of linkage editor commands.
SPRINT - printed output produced by the linkage editor.
SPUNCH - default output unit for object modules produced by the
    linkage editor.
SERCOM - linkage editor diagnostic messages.
GUSER - user responses if conversational mode is used.
```

The following parameters may be specified in the PAR field of the \$RUN command. The parameters must be separated by commas or blanks. The minimum abbreviation for each parameter is underlined.

COMSAVE/NOCOMSAVE
COMSAVE specifies that COM (comment) records are to be saved during linkedit processing. NOCOMSAVE specifies that COM records are not saved. The default is NOCOMSAVE.

GENSAVE/NOGENSAVE
GENSAVE specifies that object module generation information on COM and END records is to be preserved if a combine operation is performed. NOGENSAVE specifies that the generation information is to be discarded. The default is NOGENSAVE.

SYMSAVE/NOSYMSAVE
SYMSAVE specifies that SYM (symbol) records are to be saved during linkedit processing (except during a combine operation). NOSYMSAVE specifies that SYM records are not saved. The default is SYMSAVE.

EMPTY/NOEMPTY
EMPTY specifies that the output file assigned to SPUNCH is to be emptied before use. NOEMPTY specifies that the output file is not to be emptied. The default is NOEMPTY.

ORL=n
"n" specifies the maximum output record length to be used for output produced by the linkage editor. The default is the maximum record length of the output file or device.

GAPSIZE=n
"n" specifies the maximum object module text gap size to be filled during linkage editor processing (see the description of the GAPSIZE modifier in the subsection "Command Modifiers").

MISCSAVE/NOMISCSAVE
MISCSAVE specifies that miscellaneous loader records (ALI, DEF, LCS, NCA, OPT, and RIP) are to be preserved during linkedit processing. NOMISCSAVE specifies that these records are to be deleted. The default is MISCSAVE.

MSGSAVE/NOMSGSAVE
MSGSAVE specifies that MSG (message) records are to be saved during linkedit processing. NOMSGSAVE specifies that MSG records are not to be saved. The default is MSGSAVE.

## TERSE/VERBOSE

The TERSE/VERBOSE parameters control the amount of information produced by the verification of some commands. TERSE specifies that minimal information is requested; VERBOSE specifies that full information is desired. The default is VERBOSE. This parameter pair has no effect if NOVERIFY is specified.

VERIFY/NOVERIFY
VERIFY specifies that verification for each command is requested. NOVERIFY suppresses the verification. The default is VERIFY.

WXTOER/NOWXTOER
WXTOER specifies that all ESD (external symbol dictionary) symbols of type $W X$ (weak external reference) are to be converted to type ER (external reference) during linkedit processing. NOWXTOER specifies that type WX symbols are not converted. The default is NOWXTOER.

Those users who want only to reformat object modules can simply issue the following command:

## \$RUN *LINKEDIT SCARDS=inFDname SPUNCH=outFDname

In this case, object modules are read in from "inFDname", converted to the linkage editor internal representation, and written on "outFDname". The logical structure of the modules is completely and identically preserved, with the following exceptions:
(1) REP records encountered in input modules ${ }^{1}$ are absorbed into the text of output modules.
(2) If multiple ENT records are encountered, all but the first record encountered are discarded.
(3) Nonabsolute DEF records are incorporated into the external symbol dictionary of the appropriate output modules.
(4) The first LDT record encountered terminates input, but is preserved in the output module(s).
(5) Library control records are ignored, so if a library is read in, all its modules are written out, minus library control records.
(6) COM records are discarded.

The PAR field allows users some control over the processing done by the linkage editor. Those users who wish to use the more advanced features of the linkage editor must use the linkage editor command language described below. Commands are read from SCARDS and printed

1"Input modules" are object modules read in by the linkage editor. "Output modules" are object modules written out by the linkage editor.
output is written on SPRINT; thus, the typical \$RUN command to use the linkage editor in command mode is:
\$RUN *LINKEDIT
Command input is terminated by an end-of-file or by a STOP command.
The paragraphs which follow describe in detail the linkage editor command language. Before reading this, the reader is advised to look at the sample output at the end of this section in order to get a general idea of how the commands are used.

## LINKAGE EDITOR COMMAND LANGUAGE

General Syntax

The general form of a command is

$$
\begin{aligned}
\text { commandname } \mid \text { @modifier } \mid \ldots[\text { operand] ... } \\
\left.\left\lvert\, \begin{array}{l}
|@ \rightarrow m o d i f i e r| \\
\mid @-m o d i f i e r ~ \\
{[ }
\end{array}\right.\right]
\end{aligned}
$$

where the characters "ᄀ" and "-" are used to negate a modifier.
The following rules apply to command usage:
(1) A command starts with the first nonblank character; thus, commands need not start in column 1. There is an exception to this rule: the COMBINE command must begin in column 1 ; otherwise, the linkage editor will attempt to treat the command as a loader COM record.
(2) Specification of the command name and command modifiers should contain no embedded blanks.
(3) At least one blank should separate the command name and the first operand.
(4) Multiple operands must be separated by one or more blanks.
(5) If the last character of the current input line is a minus sign, the next input line will be taken as a continuation of the current line. The first character of the next line replaces the minus sign. There is no limit to the number of continuation lines, and the continuation character can occur anywhere within a command. Note: for batch use, the minus sign must be punched in column 80, since all 80 columns of a card are read.
(6) Any command or command modifier may be abbreviated by entering only an initial substring. The minimal initial substring allowed is underscored in each command or command modifier description.
(7) A command line beginning with an asterisk "*" is considered to be a comment and is not processed, other than possibly being echoed, by the linkage editor.
(8) A command line beginning with a dollar sign "\$" is assumed to be a MTS command and is executed via a call to the system CMD subroutine.
(9) Modifiers appended to a command are global to all of the operands of the command. Certain commands (e.g., UNLINK, DISPLAY) allow certain modifiers (e.g., CS, TYPE) to be appended to the operands of the command; these modifiers are local to the operand.
(10) A command line may be entered in upper- or lowercase. If the command is in lowercase, the entire input line is translated to uppercase. Thus, to specify a lowercase symbol in an "slist", the command must be entered in uppercase.

The following table summarizes the linkage editor commands together with their applicable modifiers.

| Command | Operand | Applicable Modifiers |
| :---: | :---: | :---: |
| $\underline{\text { ADD }}$ | [FROM] FDname [slist] | A , CHECK, COMSAVE, GAPSIZE, GENSAVE, MISCSAVE, MSGSAVE, NV, SLI, SYMSAVE,TERSE, V, VERBOSE, WXTOER |
| ALIAS | name [土xxx] alias [alias] | NV, V |
| ATTRIBUTE | [ON FDname] [symbol [,] ...] | None |
| BLAST | None | None |
| CLEAR | None | NV, V |
| COMBINE | [slist] | BC, CHECK, COMSAVE, GAPSIZE, GENSAVE,MISCSAVE, MSGSAVE, NAME, NV, TERSE, V, VERBOSE |
| COMMENT | comment | None |
| COPY | [FROM] inFDname [TO] outFDname | [slist] <br> COMSAVE, EMPTY, ENTRY=, GAPSIZE, GENSAVE, MISCSAVE, MSGSAVE, NV, ORL, SYMSAVE, TERSE, V, VERBOSE, WXTOER |
| CSECT | \{section\|entry\} | A, NV, V |
| DELETE | slist | CHECK, CSECT, ENTRY, NV, OM, TERSE, $\mathrm{V}, \mathrm{VERBOSE}$ |
| DISPLAY | object | CSECT=, LENGTH=, TYPE= |
| DUMP | [ON FDname] [slist] | None |
| INCLUDE | [FROM] FDname [slist] | CHECK, COMSAVE, GAPSIZE, GENSAVE, MISCSAVE, MSGSAVE, NV, SLI, SYMSAVE,TERSE, V,VERBOSE, WXTOER |
| LIST | [ON FDname] [object [[FOR] slist]] |  |
|  |  | CSECT, ${ }^{\text {OM }}$ |
| MAP | [ON FDname] [slist] | EMPTY,FULL, SORT |


| MCMD | MTS command | None |
| :---: | :---: | :---: |
| MODIFY | object value | $\underline{C S E C T}=, \underline{L E N G T H}=, \underline{\mathrm{NV}}, \underline{\mathrm{T}} \mathrm{YPE}=, \underline{\mathrm{V}}$ |
| MTS | [optional command] | None |
| PUNCH | [ON] FDname [slist] | CHECK, COMSAVE, ENTRY=, GAPSIZE, GENSAVE, MISCSAVE, MSGSAVE, NV, ORL, SYMSAVE, TERSE, V, VERBOSE EMPTY |
| PURGE | [slist] | FULL, NV, TERSE, V, VERBOSE |
| RENAME | old1 [=] new1 [[,]old2 [=] new2]] | NV, V |
| REPLACE | [FROM] FDname [slist] | CHECK, COMSAVE, GAPSIZE, GENSAVE, MISCSAVE, MSGSAVE, NV, SLI, SYMSAVE,TERSE,V,VERBOSE, WXTOER |
| RETURN | None | None |
| SCAN | [ \{section\|address1...address2\} | value] |
|  |  | CSECT=, LENGTH= , TYPE= |
| SET | lhs=rhs [[,] lhs=rhs] | None |
| STOP | None | None |
| UNLINK | [slist] | CSECT, NV, OM, TERSE, V, VERBOSE |
| UPDATE | [FROM] FDname [slist] | CHECK, COMSAVE, GAPSIZE, GENSAVE, MISCSAVE, MSGSAVE, NV, SLI, SYMSAVE, TERSE, V, VERBOSE, WXTOER |
| XREF | [[ON] FDname] | None |

Linkage Editor Commands

The following paragraphs describe the commands which are available together with the applicable modifiers. In what follows, "slist" stands for
\{ALLBUT|[ONLY]\} symbol [[,]symbol] ...
where the brackets [...] denote optional items and the braces \{...|...\} denote alternatives. The commands that allow modifiers to be appended to the symbols in the "slist" parameter have an additional listing of applicable modifiers termed "operand modifiers."

| Command: | ADD [FROM] FDname [slist] |
| :--- | :--- |
| Modifiers: | A, CHECK, COMSAVE, GAPSIZE, GENSAVE,MISCSAVE, MSGSAVE |
| NV, SLI, SYMSAVE, TERSE,V,VERBOSE, WXTOER |  |

Command: $\quad$ ALIAS name [ $\pm x x x]$ alias [alias] ...
Modifiers: NV,V
Example: ALIAS MAIN+2B04 PATCH
Explanation: The ALIAS command allows the specification of alternate names for control sections or entry points. The parameter "name" must be a control section or entry point name and may be specified with an optional signed hexadecimal displacement "xxx". Each "alias" symbol is inserted into the ESD (external symbol dictionary) table of the module containing "name" as a type LD item; thus, the "alias" may not be previously defined within the module. Any number of alternate names may be specified.

Command: $\underline{A T T R I B U T E ~[O N ~ F D n a m e] ~[s y m b o l ~[,] ~ . . .] ~}$
Modifiers: None
Example: ATT MAIN X Y <MAIN
Explanation: The attributes of all symbols specified are written on SPRINT unless "ON FDname" is specified, in which case they are written on the file or device "FDname". If an object module name (see the subsection "Object Module Naming Conventions") is specified in the symbol list, the attributes of the module are printed. These include:

- assigned and internal names
- number of control sections
- total control section length
- total common section length
- number of special records, if any

External symbol dictionary (ESD) names may also be specified in the symbol list. The attributes listed for an ESD name may include any or all of the following items.

- assigned and internal names of the containing module
- type
- external symbol identification (ESID)
- value
- length
- name of containing section
- alignment factor

If the symbol list is omitted, the attributes of all object modules in the linkage editor data structure are printed. Object modules that have no assigned name are identified by their internal name and the designator "<>".

| Command: | BLAST |
| :---: | :---: |
| Modifiers: | None |
| Example: | BLAST |
| Explanation: | The BLAST command causes the linkage editor to ignore the current data structure without attempting to release the data structure in the normal manner via the CLEAR command. This is generally useful only for system developmental work on the linkage editor. |
| Command: | CLEAR |
| Modifiers: | NV, V |
| Example: | CLEAR@NV |
| Explanation: | The CLEAR command completely clears out the linkage editor internal data structure. This command is intended to allow the user to manipulate independent collections of modules without having to reload the linkage editor. It also allows the user to "start over" after a mistake has been made. |


| Command: | COMBINE [slist] |
| :---: | :---: |
| Modifiers: | A, BC, CHECK, COMSAVE, GAPSIZE, GENSAVE, MISCSAVE, MSGSAVE, NAME, NV, TERSE, V, VERBOSE |
| Example: | COM@NV ALLBUT SUBR1 SUBR2 |
| Explanation: | The COMBINE command causes specified modules to be combined into a single control section module. This operation is nonreversible in the sense that modules once combined cannot subsequently be separated back into their original form(s), i.e., the "combine" operation implies a loss of certain structural information. The module which results from a "combine" operation is in optimal form; that is, both CPU time for loading by the dynamic loader and external storage requirements are minimized. |
|  | The name assigned to the new combined module is by default the same as the first control section included in the combined module. This name can be overridden by the NAME modifier or the SET NAME command. |
|  | The BC modifier may be used to bind definitions of common sections into the "combined" module. In the default case this is not done, i.e., all type CM ESD items, except labeled common sections initialized in a BLOCK DATA subprogram, are preserved. Control sections representing common blocks initialized via BLOCK DATA are indistinguishable from other control sections and, therefore, are always subject to combining. The advantage of binding common definitions is further optimization of the resultant module. The disadvantage is that the binding is irrevocable; it is therefore impossible to provide larger definitions of common blocks, to have the dynamic loader select the largest definition, or to perform any similar operations. Use of the BC modifier is recommended only for relatively static production programs. |


| Command: | COMMENT comment |
| :--- | :--- |
| Modifiers: | None |
| Example: | COMMENT - NOW WE OPTIMIZE THE LOAD TIME. |
| Explanation: | This command is simply a NOP (no operation) and is useful <br> for documenting sequences of commands given to the |
|  | linkage editor. (See the sample run below.) Note that |
|  | the command COM is an abbreviation for COMBINE, not for |
|  | COMMENT. Note that command lines which begin with an |
|  |  |


| Command: | COPY [FROM] inFDname [TO] outFDname [slist] |
| :---: | :---: |
| Modifiers: | COMSAVE, EMPTY, ENTRY=, GAPSIZE, GENSAVE,MISCSAVE, MSGSAVE, NV, ORL, SYMSAVE, TERSE, V, VERBOSE, WXTOER |
| Example: | COPY@ORL=80 FROM - LOAD TO *PUNCH* ONLY MAIN |
| Explanation: | The COPY command provides the conversational user with a facility similar to the SCARDS to SPUNCH mode of the linkage editor. The action of this command is similar to that of the following command sequence: |
|  | ```INCLUDE inFDname slist PUNCH outFDname slist DELETE slist``` |
|  | with the difference being that, in order to decrease virtual memory charges, modules are included, punched (if specified in "slist"), and deleted one at a time. Because of this difference, care should be taken when applying the ENTRY= modifier since an ENT record will be punched for each module. The use of this modifier is recommended only when a single module is copied. |
| Command: | CSECT \{section\|entry\} |
| Modifiers: | A, NV, V |
| Example: | CSECT PC\#1 |
| Explanation: | The control section specified by "section" or containing the entry point "entry" is made the active section. This command should be given before the first DISPLAY, MODIFY, or SCAN command. Private control sections can be specified by their assigned names (of the form "PC\#i), available via the ATTRIBUTE, LIST, or MAP commands. |
| Command : | DELETE slist |
| Modifiers: | A, CHECK, CSECT, ENTRY, NV, OM, TERSE, V,VERBOSE |
| Example: | DELETE SUBRA, SUBRB |
| Explanation: | Through the DELETE command, the modules implied by the symbols in "slist" are deleted from the linkage editor internal structure. The primary use of this command is to allow replacement of module definitions. (See also the REPLACE command description.) |

It is also possible to delete control sections or entry points from a module. Thus, the modifiers CSECT, ENTRY, and $O M$ are used to distinguish between the data type of the command operands. For example,

DELETE X
deletes the module defining the symbol $X$ ( $O M$ is the default in this case), while

DELETE@CSECT Y
deletes the control section $Y$ from the module that contains it, and

DELETE@ENT Z
deletes the entry point $Z$ from its containing module (this is equivalent to the command PURGE Z). Deleting a control section from a multiple control section module requires that the control section first be unlinked (see the UNLINK command description); thus, the command

DELETE@CSECT Y
is equivalent to the command sequence
UNLINK@CSECT Y
DELETE Y
The default operand type is OM.

| Command: | DISPLAY object |
| :---: | :---: |
| Modifiers: | CSECT $=$, LENGTH, TYPE |
| Operand |  |
| Modifiers: | CSECT=, LENGTH, TYPE |
| Example: | DIS@T=I 0...20 INLOOP+4 0AC@T=E |
| Explanation: | This command allows object module text to be displayed in a manner similar to the equivalent debug mode command. "object" is either a single address parameter (e.g., 100) |
|  | or a block address parameter (e.g., 100...1F0). An address parameter may be either a relative hexadecimal |
|  | address or a symbol with an optional signed hexadecimal |
|  | displacement (e.g., INLOOP+4). Only symbols defined in |
|  | the external symbol dictionary (i.e., from an ESD card) |
|  | are recognized; SYM records are not interpreted. A |
|  | hexadecimal address must begin with a decimal digit |

(0-9), i.e., "AC" is considered a symbol while "0AC" is treated as a hexadecimal number.

Each "object" is converted according to the type and length specified and printed along with a one-character code that indicates the parameter type. The type codes are defined with the description of the TYPE modifier.

| Command: | DUMP [ON FDname] [slist] |
| :---: | :---: |
| Modifiers: | None |
| Example: | DUMP ON *PRINT* |
| Explanation: | The DUMP command is used for debugging the linkage editor. It produces an edited external representation of the linkage editor internal data structure which is unintelligible to anyone but the custodians of the linkage editor. The user should use this command only when prompted to do so; it is expensive. |
| Command: | INCLUDE [FROM] FDname [slist] |
| Modifiers: | A, CHECK, COMSAVE, GAPSIZE, GENSAVE, MISCSAVE, MSGSAVE, NV, SLI, SYMSAVE,TERSE,V,VERBOSE, WXTOER |
| Example: | INCLUDE FILE1+FILE2 $(100,199)$ ALLBUT QQ |
| Explanation: | Modules are read in from the specified file or device and then converted to linkage editor internal representation. Input is terminated by an end-of-file or by the first LDT record encountered. If the COMSAVE and/or GENSAVE modifiers are not given, then any COM records or GENSAVE records encountered, respectively, are discarded. example 2 in the sample run below.) If the SLI modifier is not given, all library control records encountered are ignored, with the net effect that all library are INCLUDEd. |
| Note: | This command is the same as the ADD command. |
| Command: | LIST [ON FDname] [object [ [FOR] slist]] |
| Modifiers: | OM, CSECT |
| Operand |  |
| Modifiers: | OM, CSECT |

Example: LIST GENINFO FOR MAIN
Explanation: The LIST command allows the user to obtain information about the object modules currently in the linkage editor data structure. The items of information about the modules specified by "slist" are specified by "object", where "object" may be any one of the following items:

| OMS | list all assigned object module names with their corresponding internal names |
| :---: | :---: |
| ENTRYS | - list all defined symbols (types SD, LD, LR) |
| LDS | - list all type LD entry point symbols |
| LRS | - list all type LR entry point symbols |
| CSECTS | - list all control section names |
| PCS |  |
| PRS | - list all pseudo-register definitions |
| ERS | - list all external references |
| WXS | - list all weak external references |
| COMMOMS | - list all common definitions |
| CMS |  |

GENINFO - list END record generation information MISC - list all miscellaneous records (COM, LCS, RIP,LDT, etc.)
RLDS - list the relocation dictionary (RLD) for the specified modules. If the CSECT modifier is applied to a symbol in the "slist", then only the relocations for that control section are listed; otherwise, RLDs for each control section in the specified module are listed. The list format is similar to that of the program *OBJLIST (LIST=RLD option) except that symbol names are printed for the relocation and position pointers instead of their corresponding ESIDs.

If "slist" is not specified, the "object" information is listed for all modules currently in the linkage editor data structure. If "object" is omitted, OM type information is printed for all object modules. If "slist" is specified, "object" must also be specified. LIST information is written to SPRINT unless "ON FDname" is specified, in which case it is written to the specified file or device.

| Command: | MAP [ON FDname] [slist] |
| :---: | :---: |
| Modifiers: | A, EMPTY, FULL , SORT= \{NAME \| ESID $\}$ |
| Example: | MAP@SORT=ESID |
| Explanation: | The external symbol dictionaries of all modules specified are printed out in symbolic form. If the FULL modifier is not specified, only information about control sections, entry points, and common definitions is printed (see examples 2 and 3 in sample terminal output). If the FULL modifier is given, all information is printed, including external references and PL/I pseudo-register definitions. The use of the MAP command is especially recommended after the COMBINE command has been given; this is the only practical way to ascertain the relative placement of original modules in the COMBINEd module. If the SORT modifier is not given, MAP information is sorted by address for each module. Blank lines are used as separators between each of the module maps. Preceding each module map is a header containing the assigned name of the module, the cumulative length of all control sections within the module, and the entry point to the module. MAP output is written on SPRINT unless "ON FDname" is specified, in which case it is written on the specified file or device. |
| Command: | MCMD MTS command |
| Modifiers: | None |
| Example: | MCMD EMPTY - OBJ |
| Explanation: | The MTS command specified is executed and control is returned immediately to the linkage editor. Alternatively, the user may issue the MTS command directly in linkage editor command mode by prefixing it with a dollar sign, e.g., \$EMPTY -OBJ. See Example 2 in the sample output. |
| Command : | MODIFY object value |
| Modifiers: | CSECT = , LENGTH, NV , TYPE, V |
| Operand |  |
| Modifiers: | CSECT=, LENGTH, TYPE |
| Example: | MOD@C=MAIN 100 I'STM ECD00C,LR CF' |

Explanation: This command allows object module text to be altered in a manner similar to the equivalent debug mode command. The first parameter specifies the locations that are to be modified and the second parameter specifies the values to be used for the modification.
"object" is a single address parameter, i.e., either a valid relative hexadecimal address or a symbol with optional signed hexadecimal offset, which specifies the location to be modified.
"value" specifies a list of one or more constants delimited by commas and enclosed in primes. A duplication factor and/or type specifier may optionally prefix the "value". The duplication factor must be an unsigned decimal integer, and the type specifier may be any of the code types listed in the description of the TYPE modifier. Thus, the modification values may be given as an assemblerlike constant list (e.g., 18F'0' or CL8'SCARDS' or $E^{\prime} 1.0,2.0,3.0^{\prime}$ etc.). No modifiers may be appended to "value" itself.

Verification of the modification is given by printing both the old value and the new value of the location modified. Verification may be suppressed by appending the modifier NV to the command, or by globally turning off verification via the SET VERIFY=OFF command.

Special processing is done by the linkage editor if "value" is an adcon (address constant, e.g., A'MAIN+20' or $V^{\prime}$ SERCOM'). First, any relocations for the location specified by "object" are deleted from the relocation dictionary (RLD) tables. Thus, it is possible to delete a RLD item by modifying the appropriate text location to a nonrelocatable (i.e., absolute) address constant (e.g., $\left.A^{\prime} 0^{\prime}\right)$. If this location is the only reference to an external symbol (ER) or pseudo-register (PR), then the type $E R$ or $P R$ symbol is deleted from the external symbol dictionary (ESD) tables and a warning message is given. Then the adcon "value" is examined to determine if it requires relocation (for example, $A^{\prime} X+4-X^{\prime}$ is simply the constant $\left.A^{\prime} 4^{\prime}\right)$; if it does, relocation information is inserted into the RLD tables for this location. Finally, if "value" specifies an external address constant (type V) or a pseudo-register (type Q) that is not defined in the ESD tables, then an external reference or pseudoregister is created and inserted into the tables and a warning message is given. Currently, a pseudo-register is assigned a doubleword alignment and length.

| Command: | MTS [MTS command] |
| :--- | :--- |
| Modifiers: | None |
| Example: | MTS EMPTY MYOBJ OK |
| Explanation: | If no MTS command is given, control reverts to MTS |
|  | command mode in such a way that the linkage editor may be |
|  | reentered with a \$RESTART command. If an MTS command is |
|  | given, it is passed to the MTS command language inter- |
|  | preter in such a way that the linkage editor retains |
|  | control, i.e., no \$RESTART need subsequently be issued. |

almostalways an error on the user's part. PL/I users should never purge the symbols IHEMAIN and IHENTRY which are referenced by the PL/I library.

with the additional feature that the original ordering of the input modules is preserved. Note that the UPDATE command performs a very similar function.

| Command: | RETURN |
| :---: | :---: |
| Modifiers: | None |
| Example: | RETURN |
| Explanation: | Control returns to MTS command mode in such a way that the linkage editor may be reentered with a \$RESTART command. The command is identical to the MTS command with no operands specified. |
| Command: | SCAN [\{section\|address1...address2\} value] |
| Modifiers: | CSECT=, LENGTH, TYPE |
| Operand |  |
| Modifiers: | CSECT=, LENGTH, TYPE |
| Example: | SCAN@C=MAIN 0...3FF I'SR 00' |
| Explanation: | If "section" is specified, the linkage editor will search through the text of the named section in an attempt to find the value specified. If "address1...address2" is specified, then "address1" and "address2" are the lower and upper bounds of the text area to be searched. |
|  | "value" must be enclosed in primes and may be optionally prefixed by a type specifier (see the description of the TYPE modifier for a list of the valid type codes). No duplication factor is allowed on "value", nor may a list of constants be specified if "value" is an address constant (types A, V, and Q). No modifiers may be appended to "value" itself. |
|  | If no parameter is specified, then the search resumes, starting at the first location beyond the previous match from the previous SCAN command. |
|  | The search for the specified value is performed with respect to the appropriate boundary alignment of the value specified, e.g., instructions are scanned for on halfword boundaries, character constants on byte boundaries, etc. |

Command:

Modifiers: None
Explanation: Most of the items which can be specified in a SET command are also available as modifiers to individual commands. The SET command simply changes the global default value for such modifiers so that the same modifier values need not be given repeatedly.

If the linkage editor encounters any errors when QUIT is turned ON in batch mode, the user is signed off. QUIT defaults OFF for batch and is always OFF for conversational use.

If ECHO is turned ON, linkage editor commands are ECHOed on SPRINT. ECHO defaults to ON unless the commands are being entered directly from a terminal.

The global modifier character "@" may be changed with the MODCHAR=c item, where "c" is a single character (preferably not alphanumeric). This is necessary if it is desired to enter a symbol in a "slist" parameter that contains an embedded "@" character, as the linkage editor examines each symbol for appended modifiers.

The address relocation factor of the DISPLAY, MODIFY, and SCAN commands may be changed with the set item "RF= xxxxxx", where "xxxxxx" is a one to six character hexadecimal number. The default is RF=0. This feature is primarily useful for combined modules, when it is desired to refer to a text location in a combined control section by the original assembly listing address.

| Command: | STOP |
| :---: | :---: |
| Modifiers: | None |
| Example: | STOP |
| Explanation: | Linkage editor processing is terminated. This command is identical to the RETURN command. <br> An end-of-file in the command stream also terminates linkage editor processing. |
| Command: | UNLINK [slist] |
| Modifiers: | A, CSECT, NV, OM, TERSE, V, VERBOSE |
| Operand |  |
| Modifiers: | CSECT, OM |
| Example: | UNLINK ONLY X1,Y2@CS |
| Explanation: | The UNLINK command breaks up multiple-csect object modules into a sequence of single-csect modules. This is not the inverse of the COMBINE command which produces a single-csect module. The inverse to UNLINK, the LINK command which combines object modules without combining control sections, is currently not available. When unlinking modules which contain private (blank-named) control sections, any references to the private sections from other control sections in the same module require the private sections to be renamed to a nonblank name. In this case, a warning message is given and a name of the form "PC\#i" is generated and used. |


| Command: | UPDATE [FROM] FDname [slist] |
| :---: | :---: |
| Modifiers: | A , CHECK, COMSAVE, GAPSIZE, GENSAVE, MISCSAVE, MSGSAVE, NV, SLI, SYMSAVE,TERSE,V,VERBOSE,WXTOER |
| Example: | UPDATE FROM - LOAD |
| Explanation: | The UPDATE command reads potential replacement modules |
|  | from "FDname" and selectively replaces modules in the |
|  | linkage editor internal data structure in accordance with |
|  | "slist". Any additional modules on "FDname" which are |
|  | not part of the current internal data structure are also |
|  | included. The UPDATE command can be thought of as a |
|  | convenient way of performing the following operations: |
|  | DELETE slist |
|  | INCLUDE FROM FDname |

with the additional feature that the original ordering of the input modules is preserved. Note that the REPLACE command performs a very similar function.

Command: XREF [[ON] FDname]
Modifiers: None
Example: XREF
Explanation: For each control section in the linkage editor internal data structure, all symbols "referenced" from that control section are printed. A "reference" may refer to another control section, to a common section, or to a PL/I pseudo-register. The cross-reference listing is printed out in two formats: SYMBOL1 -> SYMBOL2 and SYMBOL1 <- SYMBOL2. In each format the referenced symbol is pointed to by the arrow. In both formats alphabetization is done according to SYMBOL1, and in the case of ties, according to SYMBOL2. XREF output is written on SPRINT unless "ON FDname" is specified, in which case it is written on the specified file or device.

Command Modifiers

The following paragraphs briefly describe the available command modifiers. More detailed descriptions of the effects of these modifiers are given in the subsection "Linkage Editor Commands."

Modifier: $\underline{A}$
Example: DEL@ A X
Explanation: A stands for ALL. The A modifier controls the action taken when a multiply-defined symbol is encountered in a "slist" parameter; this modifier means "take all definitions" (the default). The A modifier applies to the commands ADD, COMBINE, DELETE, INCLUDE, MAP, PUNCH, PURGE, RENAME, REPLACE, UPDATE, and UNLINK. For example, the command

DELETE@A X
deletes all modules that define the symbol $X$. If this modifier is negated (i.e., ᄀA), then the linkage editor will detect any multiple definitions of symbols in a "slist" parameter and request that the user specify which definition is to be used. For example, if the symbol $X$ has two definitions, a csect in one module and an entry point in another, then the command

DELETE@ ${ }^{\text {A }} \mathrm{X}$
prints the following message (the user's reply is in lowercase).
***SYMBOL "X" IS MULTIPLY DEFINED.
PLEASE INDICATE WHICH OF THE FOLLOWING DEFINITIONS
ARE INTENDED BY RESPONDING WITH "Y" OR "N". CSECT "X" IN MODULE "<X" (<OM\#1>)? Y ENTRY "X" IN MODULE "<Z" (<OM\#3>)?n

In this example, the linkage editor prints all definitions in the order of occurrence with module "<X" (internal name <OM\#1>) having been included before module "<Z" (internal name <OM\#3>). In this case, the user has indicated that module "<X" is to be deleted, but not "<Z". The global default for this modifier may be changed with the SET command, i.e.,

SET ALL= $\{O N \mid O F F\}$
The default is ON.

May 1983

| Modifier: | BC |
| :---: | :---: |
| Example: | COMBINE@BC |
| Explanation: | $B C$ stands for bind common. This modifier may be applied to the COMBINE command; BC causes definitions of common blocks to be irrevocably bound into the combined module. |
| Modifier: | CHECK |
| Example: | COMBINE@CHECK |
| Explanation: | The CHECK modifier may be applied to the COMBINE, DELETE, INCLUDE, PUNCH, REPLACE, and UPDATE commands. If the CHECK modifier is specified, a consistency check is made to check that all symbols in a module have been specified if any one symbol in the module has been specified. For example, if a module is comprised of control sections A and $B$, the following command will elicit an error message: <br> DELETE ONLY B <br> CHECK may be globally enabled by the SET command, e.g., <br> SET CHECK=ON |
| Modifier: | COMSAVE |
| Example: | INCLUDE@COMSAVE MYFILE |
| Explanation: | This modifier causes COM records to be preserved in ADD, INCLUDE, PUNCH, and COMBINE commands. The UPDATE and REPLACE commands may also be modified by COMSAVE, since both commands perform the INCLUDE operation. The default for these five commands is to ignore or "throw away" COM records. Note that COMSAVE is evaluated independently for each of the commands mentioned above, so the sequence "INCLUDE@COMSAVE ... PUNCH" does not preserve COM records for the PUNCH command; "INCLUDE@COMSAVE ... PUNCH@COMSAVE" must be specified. See also the SET command description for SET COMSAVE=ON. |


| Modifier: | CSECT |
| :---: | :---: |
| Example: | UNLINK@CS ONLY X |
| Explanation: | By default, the operands of most linkage editor commands specify object modules. Certain commands, however, perform different functions depending on the type of their operands. The CSECT modifier may be applied to the DELETE, LIST, and UNLINK commands to specify that the symbols in the "slist" parameter imply control sections, as opposed to object modules. See the descriptions of these commands for the effect of this modifier. |


| Modifier: | CSECT=section |
| :--- | :--- |
| Example: | DISPLAY@C=MAIN 100@T=E |
| Explanation: | This modifier may be applied to the DISPLAY, MODIFY, or <br> SCAN commands (and operands) to specify the control |
|  | section ("section") which the command operands refer to. |
|  | This modifier may be globally set with the CSECT command. |


| Modifier: | EMPTY |
| :---: | :---: |
| Example: | PUNCH@EMPTY OBJFILE |
| Explanation: | This modifier may be applied to the COPY, MAP, and PUNCH commands to request that the output file is to be emptied before output from the linkage editor is written to it. |
| Modifier: | ENTRY $=$ \{symbol [+displacement]\|OFF\} |
| Example: | PUNCH@ENTRY=ASMGF1 ASMG |
| Explanation: | This modifier may be applied to the COPY and PUNCH commands. If "displacement" is omitted, then an ENT record specifying "symbol" as the entry point is punched at the beginning of the object program. Entry point specification via the ENT record is given the highest priority by the dynamic loader. The symbol specified as an entry point can be overridden only by another ENT record encountered earlier by the loader. |
|  | If "displacement" is specified, then the entry point of the module containing "symbol" is inserted on the END record of the module. In this case, the entry point address is the sum of the value of "symbol" and "dis- |

placement", where "displacement" can be from one to six hexadecimal digits. The ESID (external symbol identification) of the control section containing "symbol" is also inserted in the appropriate field of the END record.

The keyword OFF may be specified to suppress the punching of an ENT record, previously specified in a "SET ENTRY= symbol" command.

| Modifier: | ENTRY |
| :--- | :--- |
| Example: | DEL@ENT Z |

Explanation: The ENTRY modifier may be applied to the DELETE command to specify that the operands of the command are entry points (as opposed to object modules). This modifier, in effect, makes the DELETE command identical to the PURGE command.

Modifier: FULL
Example: MAP@FULL
Explanation: This modifier may be applied to the MAP and RENAME commands. By default, these commands are usually only applied to a subset of the potential set of affected items. The FULL modifier causes the command to be carried out for all items which might be affected.

Modifier: GAPSIZE=n
Example: PUNCH@GAP=400 SEQFILE
Explanation: When reading in object modules, the linkage editor maintains tables which distinguish between segments of control sections for which text was received, and "holes" for which no text was received. For example, FORTRAN program variables not initialized via the DATA statement cause holes in a FORTRAN module. When executing commands such as INCLUDE, COMBINE, PUNCH, etc., the linkage editor will fill in holes of size $\leq$ GAPSIZE with a fill character, which defaults to $\mathrm{X}^{\prime} 81$ ', but may be userspecified as two hexadecimal digits, e.g., SET FILL=00. Each hole of size > GAPSIZE will force generation of a new TXT/CSI output record when PUNCHing an object module. By filling in small gaps, the number of output records can often be greatly reduced. GAPSIZE defaults to ORL/2

| Modifier: | GENSAVE |
| :---: | :---: |
| Example: | COMBINE@GENSAVE |
| Explanation: | Many translators include generation information on the |
|  | END record of each object module produced specifying such |
|  | things as the name and version of the translator and the |
|  | time and date of translation. This information is lost during a COMBINE operation unless GENSAVE is specified. |
|  | If the PUNCH command is subsequently specified with GENSAVE, this information is punched on special COM |
|  | records at the beginning of an object module. The number |
|  | of GENSAVE items punched on a COM record depends on the |
|  | value of ORL (see the ORL modifier). If the GENSAVE |
|  | modifier is specified on the ADD, INCLUDE, REPLACE, or |
|  | UPDATE commands, the linkage editor preserves generation |
|  | information contained on any special COM records encoun- |
|  | tered in the input modules. Note that the GENSAVE and |
|  | COMSAVE modifiers are completely independent. See also |
|  | the SET command for SET GENSAVE= SON $^{\text {OFF }}$ O |

Modifier: LENGTH=i
Example: DISPLAY@LEN=16 MAIN+10
Explanation: This modifier may be applied to the DISPLAY, MODIFY, and SCAN commands (or command operands) to set the length attribute to "i", where "i" is an unsigned decimal integer. The default for input/output conversions is 4.

```
Modifier: MISCSAVE
Example: INCLUDE@MISCSAVE PGM.O
Explanation: This modifier causes miscellaneous records (ALI, DEF,
    LCS, NCA, OPT, and RIP) to be preserved in ADD, COMBINE,
    COPY, INCLUDE, PUNCH, REPLACE, and UPDATE commands. By
    default, miscellaneous records are preserved. See also
    the SET command description for SET MISCSAVE=ON.
```

May 1983

| Modifier: | MSGSAVE |
| :---: | :---: |
| Example: | INCLUDE@ ${ }^{\text {MSG }}$ USERS |
| Explanation: | This modifier causes MSG (message) records to be preserved in ADD, COMBINE, COPY, INCLUDE, PUNCH, REPLACE, and UPDATE commands. By default, MSG records are preserved. See also the SET command description for SET MSGSAVE=ON . |
| Modifier: | NAME=symbol |
| Example: | COMBINE@NAME=TEST1 |
| Explanation: | The result of a COMBINE command is a single control section module whose name is by default the same as the first control section included in the combined module. This name can be overridden by the NAME modifier or the SET NAME command. |
| Modifier: | NV |
| Example: | INCLUDE@NV -LOAD |
| Explanation: | Many of the linkage editor commands produce large amounts of verification output. NV stands for no verification and is used to suppress this output. |
| Modifier: | OM |
| Example: | UNLINK@CS X Y@OM Z |
| Explanation: | OM stands for object module. The OM modifier may be applied to the commands DELETE, LIST, and UNLINK, and to the operands of the LIST and UNLINK commands to specify that the operands refer to object modules (the default). In the above example, the symbols $X$ and $Z$ specify control sections, while $Y$ specifies an object module. See the descriptions of these commands for the effect of this modifier. |


| Modifier: | ORL=n |
| :--- | :--- |
| Example: | PUNCH@ORL=80 LINEFILE |
| Explanation: |  |
|  | ORL stands for output record length and defines the |
| maximum record size the linkage editor will write for a |  |
| PUNCH command. If ORL is not specified, the default is |  |
| the maximum record length that can be written on the |  |
| output file or device specified in the punch command. |  |

May 1983

| Modifier: | SYMSAVE |
| :---: | :---: |
| Example: | INCLUDE@SYMSAVE MYFILE |
| Explanation: | SYMSAVE causes SYM records to be preserved in INCLUDE, PUNCH, UPDATE, and REPLACE commands. The UPDATE and REPLACE commands may be modified by SYMSAVE since both commands perform the INCLUDE operation. By default, SYM records are preserved by these five commands. Note that SYM records currently cannot be preserved by a COMBINE command. See also the SET command description for SET SYMSAVE=ON . |
| Modifier: | TERSE |
| Example: | INCLUDE@TERSE - LOAD |
| Explanation: | The TERSE modifier may be applied to some commands to abbreviate the information produced for verification. By default, full information is given to verify a command. VERBOSE is an antonym of TERSE. See also the SET command description for SET TERSE=ON. |
| Modifier: | $\underline{\text { TYPE }}=$ code [Li] |
| Example: | DIS MAIN@T=CL12 |
| Explanation: | This modifier may be applied to the DISPLAY, MODIFY, and SCAN commands (or operands) to set the type attribute to "code", where "code" is any of the single-character type-codes defined below. The default is X . |
|  | Code Type |
|  | A A-type address constant |
|  | B binary |
|  | C character (EBCDIC) |
|  | D floating-point (long) |
|  | E floating-point (short) |
|  | F fixed-point (fullword) |
|  | H fixed-point (halfword) |
|  | I machine instruction |
|  | P packed decimal |
|  | Q Q-type address constant (pseudo-register) |
|  | S S-type address constant (output only) |
|  | V V-type address constant |
|  | X hexadecimal |
|  | Y Y-type address constant <br> Z zoned decimal |

A length attribute "Li", where "i" is an unsigned decimal integer, may be appended to the single-character "code" to change the default length used in conversion. The effect of this is identical to applying the LENGTH=i modifier. Complete details on input and output conversions performed by the linkage editor are given in the subsections "Input Conversions" and "Output Conversions."

| Modifier: | $\underline{V}$ |
| :--- | :--- |
| Example: | PURGE@V |
| Explanation: | V stands for verify. If verification has been turned off <br> globally via SET VERIFY=OFF, then it can be turned on for <br> a particular command via the V modifier. |
| Modifier: | VERBOSE |
| Example: | INCLUDE@VERBOSE FROM -LOAD |
| Explanation: | If TERSE has been enabled globally via SET TERSE=ON, the |
|  | full information for the verification of a particular |
|  | Command can be produced via the VERBOSE modifier. <br> BOSE is an antonym of TERSE. See also the SET command <br> description for SET VERBOSE=ON. This modifier has no <br> effect if verification is suppressed via the NV modifier |
| or the command SET VERIFY=OFF. |  |


| Modifier: | WXTOER |
| :--- | :--- |
| Example: | INCLUDE@WXTOER FILE.O |
| Explanation: | This modifier may be applied to the ADD, INCLUDE, COPY, <br>  <br>  <br> REPLACE, and UPDATE commands to request the linkage <br>  <br> editor to convert all ESD symbols of type wX (weak <br>  <br> external reference) to type ER (strong external refer- <br>  <br> ence). By default, symbols of type wX are not converted. <br>  <br> See also the SET command description for SET wXTOER=ON. |

## LINKAGE EDITOR EXAMPLE

A sample terminal run is given below to illustrate several of the features of the linkage editor. In this example, user input is in lowercase while linkage editor output is in uppercase. Note also that the linkage editor uses an asterisk as a prompting character.
\#\$run *linkedit \#EXECUTION BEGINS
VERSION(12-16-75) 15:10:14 01-15-76
*comment - example \#1 converts object modules to card images.
*include *time
*** WARNING: INPUT STREAM TERMINATED BY AN LDT CARD
INCLUDED:
TIME
*punch@orl=80 -cardfile
PUNCHED:
TIME
*clear
DONE.
*comment - example \#2 converts object modules to optimal form.
*set symsave=off
*incl ftnobj
*** WARNING: SYM/FTN CARDS ARE BEING IGNORED.
INCLUDED:
MAIN INSUB OUTSUB
*list commons
<INSUB: BLK
*combine@bc
COMBINED MODULE:
MAIN OUTSUB INSUB BLK
*map
EXTERNAL SYMBOL DEFINITIONS: (NAME TYPE ESID ADDR VALUE AF)
<MAIN: ENTRY = MAIN SIZE = 000A10
MAIN SD 000100000 000A10 INSUB LD 0002100001
OUTSUB LD 0006D0 0001 BLK 0009 D8 0001
*purge
PURGED:
BLK INSUB OUTSUB
*map
EXTERNAL SYMBOL DEFINITIONS: (NAME TYPE ESID ADDR VALUE AF)
<MAIN: ENTRY = MAIN SIZE = 000A10
MAIN SD 0001000000 000A10
*\$cre -seqobj type=seq
\#\$CRE - SEQOBJ TYPE=SEQ
\# FILE "-SEQOBJ" HAS BEEN CREATED.
*punch -seqobj
PUNCHED:
MAIN

```
*clear
    DONE.
*comment - example #3 illustrates object module editing.
*i@nv ftnobj
    *** WARNING: SYM/FTN CARDS ARE BEING IGNORED.
*xref
        INSUB -> BLK BLK <- INSUB
        INSUB -> ERROR BLK <- MAIN
        INSUB -> FTNCMD BLKLTR <- OUTSUB
        INSUB -> IBCOM# ERROR <- INSUB
        INSUB -> OUTSUB FTNCMD <- INSUB
        MAIN -> BLK IBCOM# <- INSUB
        MAIN -> INSUB IBCOM# <- OUTSUB
        OUTSUB -> BLKLTR INSUB <- MAIN
        OUTSUB -> IBCOM# OUTSUB <- INSUB
*replace from -load only main
    *** WARNING: SYM/FTN CARDS ARE BEING IGNORED.
    INCLUDED:
        MAIN
    DELETED:
        MAIN
*update from -load allbut main
    *** WARNING: SYM/FTN CARDS ARE BEING IGNORED.
    INCLUDED:
        MAIN INSUB OUTSUB TIME
    DELETED:
        INSUB OUTSUB MAIN
*comment - the "main" deleted is from -load
*map
    EXTERNAL SYMBOL DEFINITIONS:(NAME TYPE ESID ADDR VALUE AF)
            <MAIN: ENTRY = MAIN SIZE = 000210
        MAIN SD 0001 000000 000210
            <INSUB: ENTRY = INSUB SIZE = 0004C0
        INSUB SD 0001 000000 0004C0 BLK CM 0002 000000 000038
            <OUTSUB: ENTRY = OUTSUB SIZE = 000308
        OUTSUB SD 0001 000000 000308
            <TIME: ENTRY = TIME SIZE = 0000E8
        TIME SD 0001 000000 0000E8
*delete time
    DELETED:
        TIME
* comb
    COMBINED MODULE:
        MAIN OUTSUB INSUB
*purge
    PURGED:
        INSUB OUTSUB
```

```
*rename main=prog
    OLD: "MAIN" NEW: "PROG"
*attr prog
        PROG: OM=<PROG ID=<OM#14> TYPE=SD VALUE=000000 LEN=0009D8 ESID=01
*clear
    DONE.
*comment - example #4 demonstrates the unlink command
*inc unlk.test.o
    INCLUDED:
        PC#1 L0 C1 L1 C2 L2
*list csects
        <L0: C1 C2
*map
    EXTERNAL SYMBOL DEFINITIONS:(NAME TYPE ESID ADDR VALUE AF)
```

            <LO: ENTRY = L2 SIZE = 000068
        PC\#1 PC 0001000000000028 CM 000A 000000 000010
        L0 LD \(0000200001 \quad\) C1 \(\quad\) SD 0006000028000028
        L1 LD 0000480006 C2 SD 0008000050000018
        L2 LD 0000640008
    *unlink@cs c1
*** WARNING: THE SYMBOL PC\#1 HAS BEEN DEFINED.
UNLINKED:
C1 L1
*map
EXTERNAL SYMBOL DEFINITIONS: (NAME TYPE ESID ADDR VALUE AF)
<L0: ENTRY = L2 SIZE = 000040
PC\#1 SD 0001000000000028 CM 000A 000000 000010
L0 LD $0000200001 \quad$ C2 $\quad$ SD 0008000050000018
L2 LD 0000640008
<C1: ENTRY = C1 SIZE $=000028$
C1 SD $0001000000000028 \quad$ CM 0008000000000010
L1 LD 0000200001
*clear
DONE.
*include unlk.test.o
INCLUDED:
$\begin{array}{llllll}\text { PC\#1 } & \text { L0 } & \text { C1 } & \text { L1 }\end{array}$
*unlink <lo
*** WARNING: THE SYMBOL PC\#1 HAS BEEN DEFINED.
UNLINKED:
$\begin{array}{llllll}\text { PC\#1 L0 } & \text { L1 } & \text { L1 }\end{array}$
*map
EXTERNAL SYMBOL DEFINITIONS: (NAME TYPE ESID ADDR VALUE AF)
<PC\#1: ENTRY = PC\#1 SIZE $=000028$
PC\#1 SD 0001000000000028 CM 0008000000000010
L0 LD 0000200001
<C1: ENTRY = C1 SIZE $=000028$
C1 SD 0001000000000028 CM 0008000000000010


## PRINCIPLES OF OPERATION

The discussion that follows describes the principles of operation of the linkage editor. The reader is assumed to have a working knowledge of the basic structure of object modules. Those who do not should first read the "Elementary Loader Topics" subsection in the section "The Dynamic Loader" in this volume prior to proceeding with the following paragraphs.

The operations of the linkage editor are comprised of three basic activities:
(1) Object modules are read from an external storage medium, e.g., a file or card deck, and converted to their linkage editor internal representations in virtual memory.
(2) The linkage editor internal representations of object modules may be interrogated or modified.
(3) The internal representations of object modules are converted to an appropriate external format and are written on an external storage medium.

It is important to note that complete conversion to and from linkage editor internal representation is always done in any processing of object modules, even in the case of the "SCARDS to SPUNCH" operation described earlier. This implies that certain physical transformations (optimizations) are always made in processing object modules even when no logical changes are made. It is also important to note that under no circumstances does the linkage editor make direct changes to the external representations of object modules. Changes can be made directly only to the linkage editor internal representations.

Step (1) above is usually performed via the linkage editor INCLUDE command. Step (2) is performed by a number of commands, e.g., MAP, COMBINE, PURGE, etc. Step (3) is performed by the PUNCH command. Thus, a typical sequence of linkage editor commands might be:

INCLUDE FDname
COMBINE
MAP
PURGE
PUNCH FDname

## Optimization of Object Modules

The physical format of object modules produced by compilers is often dictated by convenience and compiler organization. Seldom are such modules optimized either with respect to loading time or file storage requirements. The basic compiler-generated loader records consist of a 16-byte header and a variable-length field which can be up to 240 bytes long. However, most compilers produce card-image object modules with an average record length considerably less than 80 bytes. Hence, a very simple but effective way to reduce both the file storage requirements and loading time is to reduce the total number of records by increasing the average record length, i.e., by making the object module as compact as possible. Object modules produced by *PL1 are particularly noncompact. An 8655-line PL/I test program consisting of eleven compilations, for example, was compiled into an object module with 8063 records. By using the linkage editor to reformat the module into maximum size records, this module was compressed into 981 records in a line file. The effect on the loading time (on an Amdahl 470/V7) and file storage requirements was as follows:

Load Time Storage

| PL/I module: | 1.296 seconds | 182 pages |
| :--- | :--- | :--- |
| Compressed into line file: | 0.494 seconds | 116 pages |

A second, and more complex, way of reducing the loading time and file storage requirements is to combine into one module a collection of object modules which are always loaded together. This can be very effective for programs which consist of a large number of subprograms which were written and compiled independently for debugging purposes, and are now reliable enough to be heavily used. It should be noted, however, that this process is nonreversible and SYM records are not retained. No information is retained concerning a module's previously independent status. As an example of this further optimization, consider these eleven independent modules which were combined into 145 records in a line file by the linkage editor:

Load Time
0.328 seconds

Storage

69 pages

## Optimizations Automatically Performed

The following paragraphs describe transformations (and associated optimizations) made in the process of converting object modules to and from linkage editor internal representations.

ESD Records:


#### Abstract

ESD items are stored internally in order of their ESID numbers. An LD (entry point) item, which has no ESID, is chained to the $S D$ or $P C$ (control section) ESD item for the control section in which it is contained. On output, ESD items are written in ESID order, with LD items written immediately after the associated SD item. As many ESD items are written, per output record, as the output file or device will allow, up to a maximum of 15. ESIDs are preserved exactly, not taking advantage of the (rare) possibility of renumbering to eliminate unused ESID numbers. This strategy was adopted so that user-generated loader records which require specification of an ESID can be generated in a consistent manner both before and after simple (i.e., not using a COMBINE command) linkage editor processing.


TXT/CSI Records:
TXT and CSI records are read into an internal buffer allocated for their associated control sections. For each control section, the linkage editor maintains a sorted table which distinguishes between control section segments for which text was received and "holes" for which there is no text. The linkage editor automatically fills holes of size $\leq$ GAPSIZE with a fill character (default is X'81'). By filling in small holes, the number of TXT and CSI records often can be greatly reduced without increasing file-storage requirements. TXT and CSI records are always written in address order, using the sorted table. The length of a TXT or CSI output record is subject to three limitations:
(1) The maximum length allowable for the output file or device, e.g., 80 bytes is the upper bound for all output records to the card punch.
(2) If the gap between one block of text and the next exceeds GAPSIZE (GAPSIZE defaults to 256 or one half the maximum output file or device record length, whichever is smaller), a new record is started for the "next" block. Otherwise, the gap is filled with the fill character.
(3) Since the text of a TXT or CSI record is associated with exactly one ESID, the text corresponds to exactly one control section; hence, a TXT or CSI record can be no longer than the longest control section to be written. This means that the benefits of TXT/CSI optimization may be limited when writing large numbers of small control sections. The COMBINE command, described below, is intended to eliminate this problem, among others.

RLD Records:
Each item on a RLD record has two ESIDs associated with it: a position pointer identifying the control section containing the location-dependent data, and a relocation pointer identifying the symbol on which the value of the location-dependent data depends.


#### Abstract

In the linkage editor internal representation, RLD items are stored in order of their relocation pointers on lists attached to the control section identified by their position pointers, i.e., RLD items are maintained fully sorted by pointers. The definition of an RLD data item on an RLD record can take two forms, a short form (4 bytes) and a long form (8 bytes). The short form is used only in cases where an RLD data item has the same relocation and position pointers as the previous RLD data item in the same RLD record. In such cases, a special bit in the RLD data item is used to signal that the relocation and position ESIDs of the succeeding RLD data items are the same as those of the current item and are omitted from the succeeding item. Since the linkage editor stores (and writes) RLD items sorted by pointers, it achieves maximal compression available via the short form. On output, as many RLD items are written, per output record, as the file or device will allow, up to a maximum of 240 bytes.


Optimizations Performed by the COMBINE Command

The optimizations described above are always performed automatically. However, greater optimizations are often attainable through the use of the COMBINE command. The COMBINE command basically combines object modules containing many control sections into a single object module comprised of exactly one control section. The transformation and optimizations performed in this process are described in the paragraphs which follow.

ESD Records:
The module produced by combining object modules contains exactly one type SD (control section) ESD item. The name of the $S D$ item is the name of the first nonblank control section definition encountered in the modules being combined. The length of the combined control section $S D$ item is the sum of the lengths of all the control sections being combined, each rounded up to a multiple of 8 bytes. Control sections other than the first are changed, in the order in which they are encountered, into entry points (type LD) located at the next available displacement in the combined control section. All entry points (types LD and LR) in control sections being combined are changed into type LD entry points defined as displacements from the beginning of the combined control section.

External references from one control section to another control section are resolved if these control sections are being combined. The type ER or WX ESD items for such resolved references are deleted. This is made possible since there is only one resultant control section and such references are internal to this control section.

ESD items of types not described above (types $C M$ and $P R$, and "truly external" ER and WX types) are preserved in the combined module, but their ESIDs are renumbered to produce a module containing only contiguously assigned ESIDs.

Evidence of the changes made to ESD information may be obtained by the use of the MAP command before and after the COMBINE command is used.

## TXT/CSI Records:

All TXT and CSI records of a combined module have ESID 0001, the ESID of the resultant control section. The sorted text tables for combined control sections are merged into a single sorted text table. This means that the addresses from the original TXT and CSI records must be changed to reflect the positions of the containing control sections in the combined control section.

RLD Records:
The relocation and position pointers for all RLD items are changed to correspond to ESD items in the combined module. The displacement fields of the RLD items are changed to reflect the displacement of their original containing control section in the combined control section. Finally, the combined control section text referenced by RLD items having relocation pointer 0001 is modified to reflect the displacements of the original control sections in the new combined control section.

## Optimizations Performed by the PURGE Command

The PURGE command deletes type LD and LR ESD items from all modules in the linkage editor internal representation. Deletion of these items yields savings in loading time and a modest saving in external storage requirements. One word of warning is in order: purging entry points only makes sense when the purged entry points are unnecessary entry points in a combined module. Unnecessary entry points to a module are entry points that will never be referenced by another module. It is often the case that a combined module is self-contained, and thus the PURGE command is often used immediately after the COMBINE command. PL/I users should note that the symbols IHENTRY and IHEMAIN should never be purged. This can be achieved by the use of the ALLBUT option, e.g.,

PURGE ALLBUT IHEMAIN IHENTRY

Optimizations Obtained by the SLI Modifier

Many object modules have the property that their loading forces the loading of a small number of small modules from a large module library. A certain amount of CPU time is required to extract these modules from the library each time the program is loaded. The SLI modifier allows library modules to be extracted from a library at link-edit time. Thus, at the expense of linkage editor CPU time and increased external storage requirements, loading CPU time can be saved. See the words of warning in the SLI modifier description.

Optimizations Obtained by the BC Modifier

The BC modifier may be applied to the COMBINE command to obtain optimization associated with the processing of common sections. The BC modifier forces the linkage editor to bind into the combined module definitions of common sections that would otherwise require storage allocation and other processing when loaded by the dynamic loader. The only disadvantage of doing this is that if other modules are to be loaded along with the combined module, they must use the common definitions (lengths) bound into the combined module. In particular, longer lengths cannot be defined.

PROCESSING SPECIAL RECORDS

One of the most difficult problems facing the linkage editor is what to do with the special records which may appear in an object module file. Special records are those loader records which are generated by hand (e.g., REP, ENT, LDT) or library control records generated by *OBJUTIL (e.g., DIR, RIP, LIB). Special records pose a problem because it is difficult to ascertain which (if any) object modules are affected by these special records. It is important to note that the order in which special records appear can be crucial. If it is necessary to delete, replace, or combine object modules using the linkage editor, the problem of deleting, replacing, or combining any associated special records becomes important. This section describes the conventions used by the linkage editor to handle special records. Users who only linkedit object modules produced by compilers can ignore this section completely.

For lack of a better convention, the linkage editor considers any special records which precede or are part of an object module to be associated with that object module. That is, everything between END records is considered to be part of the same object module. This means that any operations which delete, replace, or combine object modules will also affect those special records which appear before, or within,
the object modules. The process of combining object modules with special records creates an additional problem. Because of this process, special records which originally appeared between object modules must now be moved to a new position relative to the combined module. The conventions for solving these problems are described below.

## ALI Records

If MISCSAVE is on, ALI records are preserved in order of occurrence. If combining is performed, the associated ALI records will precede the combined module.

COM Records
If COMSAVE is ON, COM records are preserved in order of occurrence. If combining is performed, the associated COM records will precede the combined module.

DEF Records
There are two types of DEF records: absolute (ESID=0) and relative (ESID>0). Relative DEF records are incorporated into the external symbol dictionary during INCLUDE processing and thus lose the properties of special records. Absolute DEF records are preserved in order and are moved to the front of combined modules, if MISCSAVE is ON.

DIR Records
DIR records are library control records. If SLI is ON, DIR records are interpreted. If SLI is OFF, DIR records are ignored. In either case, they are never preserved as part of an object module.

ENT Records
ENT records are preserved in order. If combining is performed, only the first ENT record is retained, since the loader will only look at the first one.

LCS Records
If MISCSAVE is ON, LCS records are preserved in order and are positioned after combined modules.

LDT Records
An LDT record terminates INCLUDE processing. LDT records are preserved in order. If combining is performed, only the first LDT record is retained and is positioned after combined modules.

May 1983

LIB Records

LIB records are library control cards. If SLI is ON, LIB records are interpreted. If SLI is OFF, LIB records are ignored. In no case are they preserved as part of an object module.

MDL Records
Currently, all MDL records are ignored.

MSG Records
If MSGSAVE is ON, MSG records are preserved in order and are positioned before combined modules.

NCA Records
If MSCSAVE is ON, NCA records are preserved in order and moved to the front of combined modules.

OPT Records
If MISCSAVE is ON, OPT records are preserved in order of occurrence and are positioned before combined modules.

REP Records
REP records are incorporated into the control section text during INCLUDE processing. They are never produced as output from the linkage editor.

RIP Records
There are two types of RIP records: absolute and conditional RIP records. Conditional RIP records are library control records. If SLI is ON, they are interpreted. If SLI is OFF, conditional RIP records are ignored. In no case are they preserved as part of an object module. Absolute RIP records are preserved in order and are positioned after combined modules, if MISCSAVE is ON.

ATTENTION-INTERRUPT PROCESSING

Attention interrupts within the linkage editor are processed as follows:
(1) If an attention interrupt occurs during output verification, the remainder of the verification is suppressed.
(2) If an attention interrupt occurs during the processing of a command, the command is aborted if possible. For example, an
attention interrupt during an INCLUDE command causes the command to be aborted with nothing included; an attention interrupt during a COMBINE command is not allowed; an attention interrupt during a PUNCH command causes the command to be aborted, resulting in an incomplete object file.

At any time, two successive attention interrupts without an intervening return to linkage editor command mode causes a return to MTS with the linkage editor still loaded. In this case, the linkage editor may be restarted via the \$RESTART command.

## INPUT CONVERSION

Input conversion is used with the MODIFY and SCAN commands. The MODIFY command requires two parameters:
(1) an address parameter specifying what locations are to be modified, and
(2) a list of one or more constants delimited with commas; the entire list is enclosed in primes. The constants are converted according to the type and length attributes specified.

The SCAN command also requires two parameters initially:
(1) a section name or block parameter specifying a region to be scanned, and
(2) a list of one or more constants delimited with commas; the entire list is enclosed in primes. The constants are converted according to the type and length attributes specified.

No modifiers may be appended to the constants themselves. Type and length modifiers may be appended to the command or to the first parameter. A type specifier may optionally prefix the constant list. Conventions for the representation of the different types of linkage editor constants are presented below. The total length of all constants specified in an input constant list may not exceed 256 bytes. If an explicit length is not specified via the LENGTH modifier or $L$ attribute or implied by the type attribute, the implied length of the constant is taken as the number of bytes necessary to contain the constant. The term "parameter" refers to the operand that is being modified or scanned for. An error comment is produced if an unsupported type is specified, or a constant is specified incorrectly.

A-type adcon If the specified length is one or two bytes, the constant is treated as a fixed-point decimal integer of length one or two, respectively. If the specified length is three or four bytes, the constant may be any absolute or relocatable expresssion; the only operators recognized are "+" and "-". Symbol names given in the expression must be defined within the
module that contains the current active section (specified via the CSECT command or CSECT= modifier). Special processing is done by the linkage editor if the constant expression is relocatable; see the explanations of the MODIFY and SCAN commands for details. The default length for an A-type address constant is 4 bytes. The example below modifies the location $9 B 0$ to contain the value of the symbol MAIN plus the hexadecimal displacement 1A0, and creates a RLD item for this location after deleting any previous relocations from the relocation dictionary tables.

MODIFY 9B0 A'MAIN+1A0'
A binary constant is written as a string of 1's and 0's. The implied length of a binary constant is the number of bytes (eight binary digits per byte) occupied by the constant including any padding necessary. Padding or truncation required by an explicit length specification takes place on the left. The padding bit is a 0 . In the example below, the byte specified by the relative address 100 is modified to the bit string 00000101.

MODIFY 100 B'101'
Any of the valid 256 punch combinations may be designated in a character constant. Only one character constant may be specified in the second operand to the MODIFY or SCAN command. Since multiple constants within the second operand are separated by commas, an attempt to specify two character constants results in interpreting the comma as a character. Special attention must be given to representing primes as characters. Each single prime desired as a character in the constant must be represented as a pair of primes. The maximum length of a character constant is 256 bytes. Double primes are counted as one character. If an explicit length is specified, padding or truncation will take place on the right if necessary. The padding character is a blank. In the example below, the location 2020 is modified to contain the character string ABCDE.

MODIFY $2020 C^{\prime} A B C D E^{\prime}$
A fixed-point constant consists of a signed or unsigned decimal integer. No decimal point is allowed. The length of the constant depends on the type specifier; if the type is "F", the length is 4 bytes, if "H", the length is 2 bytes. Any explicit
length specification is ignored. The example below modifies the locations 10 and 14 to the constants 1 and 2, respectively.

MODIFY $10 \mathrm{~F}^{\prime} 1,2^{\prime}$
A floating-point constant is written as a decimal number. As an option, a decimal exponent may follow. The number may be an integer, a fraction, or a mixed number. The length of the constant depends on the type specifier; if the type is "E", the constant is treated as a single-precision constant and occupies 4 bytes, if the type is "D", the constant is treated as a double-precision constant and occupies 8 bytes. Any explicit length specification is ignored. The format of the constant is as follows:
(1) The number is written as a signed or unsigned decimal value. The decimal point can be placed before, within, or after the number. If it is omitted, the number is assumed to be an integer. A positive number is assumed if an unsigned constant is specified.
(2) The exponent is optional. If specified, it is written immediately after the number as En, where "n" is an optionally signed decimal value specifying the exponent of the factor 10 . The value of the constant may be in the range of $.723700515 \mathrm{E}+76$ to .539760535E-78. If an unsigned exponent is specified, a plus sign is assumed.

In the following example, the parameter DATA is modified to 46.415.

MODIFY DATA@T=E '+.46415E2'
Hexadecimal
A hexadecimal constant consists of one or more of the hexadecimal digits $0-9$ and A-F. Constants that contain an even number of digits are translated as one byte per pair of digits. If an odd number of digits is specified, the leftmost byte has its leftmost four bits filled with hexadecimal zeros, while the rightmost four bits contain the first digit. Hexadecimal constants are right-justified. Padding or truncation required by an explicit length specification is made on the left. The padding character is hexadecimal zero. In the example below, the locations 40 and 42 are modified to the hexadecimal constants 0008 and 000F, respectively.

Instruction

Packed \& zoned

Q-type adcon

MODIFY 40 XL2'8, ${ }^{\prime}$
An "instruction" constant consists of a 360/370assembler mnemonic and a hexadecimal operand of the appropriate length. The operand is separated from the mnemonic by one or more blanks; blanks may be included within the operand, but not the mnemonic. Extended mnemonics (such as BNM) may be used, in which case the extended mnemonic will include the mask digit of the $B C$ or $B C R$ instruction. In the first example below, the location specified by the parameter MAIN is modified to the assembler instruction "STM 14,12,12(13)".

MODIFY MAIN I'STM ECDOOC'
MODIFY 240 I'BL 0 B098'
A packed or zoned decimal constant is written as a signed or unsigned decimal value. If the sign is omitted, a plus sign is assumed. The existence of a decimal point in no way affects the conversion of a decimal constant. In effect, the decimal point is ignored. If zoned decimal conversion is being performed, each decimal digit is translated into one byte. The rightmost byte contains the sign as well as the rightmost digit. For packed decimal conversion, each pair of decimal digits is packed into one byte. If an even number of packed decimal digits is specified, the leftmost four bits in the leftmost byte are set to zero and the rightmost four bits contain the first digit. No more than 16 digits may be specified in a packed decimal constant. Padding or truncation as required by an explicit length specification is performed on the left. For packed conversion, the padding character is hexadecimal 00; for zoned conversion, the padding character is the character "0" (hexadecimal FO). Examples:

```
MODIFY 100 P'23'
MODIFY 104 Z'-44'
```

A Q-type address constant is used to specify an external dummy section, i.e., pseudo-register. The length of the constant may be from one to four bytes; the default is four bytes. The constant is specified as a relocatable symbol which names a pseudo-register. For the SCAN command, this pseudoregister must be defined in the module that contains the active control section (via the CSECT command or CSECT= modifier). For the MODIFY command, this symbol need not specify a previously defined pseudoregister; in this case, a pseudo-register definition is created and inserted into the RLD tables. Cur-

|  | rently, the pseudo-register is given a doubleword <br> length and alignment. Example: |
| :--- | :--- |
| MODIFY MAIN+22 QL2'MAINB' |  |

OUTPUT CONVERSION

Output conversion is required for the DISPLAY command and in verification for the MODIFY command. All of the conversion types listed with the description of the TYPE modifier are supported by the linkage editor output conversion routines. The following conventions are used when the linkage editor attempts to display illegal data.

Character If a hexadecimal code is encountered which has no character equivalent, a question mark is substituted.

Instruction If the opcode field of an instruction does not correspond to a machine operation, the instruction is printed as two hexadecimal digits surrounded by asterisks. The instruction is assumed to be 2 bytes in length ( $R$ R type).

Packed \& zoned If the data are not in the packed or zoned format, the number is printed in hexadecimal format.

## OBJECT-MODULE NAMING CONVENTIONS

Most linkage editor commands require the specification of one or more object modules. However, object modules do not have translator-assigned names. Thus, a problem exists in how to specify a particular object module at the command language level. The usual approach is to specify a module by implication, e.g.,

DELETE X
means "delete the module which defines X". Thus, the object module is specified by entering a symbol defined within the module in the "slist" parameter. However, this method has at least three shortcomings:
(1) A module may consist of exactly one private control section and hence, define no symbols.
(2) The symbol $X$ may be multiply-defined.
(3) There is a problem in how to specify a particular item within a module, e.g.,

DELETE X
can never mean delete the control section $X$ from its containing module.

The problem of distinguishing the data type of an operand is generally handled by modifiers with appropriate defaults. Thus, (3) above is resolved by appropriate use of the modifiers OM, CSECT, and ENTRY, described in the subsection "Command Modifiers."

Solutions to (1) and (2) above, however, require alternate ways of specifying object modules. For these reasons, the linkage editor assigns a name to an object module of the form "<sym", where "sym" is the name of the first symbol defined in the module if there is one; otherwise, the object module name remains undefined. This assignment is made during INCLUDE processing and can be changed via the RENAME command. The assigned name can then be used to specify a module as follows:

DELETE < X
deletes the module named $<\mathrm{X}$ which defines the symbol X .
In addition, the linkage editor assigns an internal name to each object module. This name is also assigned during INCLUDE processing and has the form "<OM\#j>", where j=1,2,.... This internal name, although artificial, makes it possible to unambiguously specify an object module.

The LIST and ATTRIBUTE commands (described in the subsection "Linkage Editor Commands") are available to let the user know what the assigned

MTS 5: System Services
and internal name of an object module is in order to reference it. Thus, a module containing a private control section may be referenced by its internal name, as shown in the example below.
*LIST PCS
<OM\#1>: PC\#1
*DELETE <OM\#1>
DELETED:
PC\#1

THE OBJECT-FILE EDITOR

The MTS object-file editor is used for the editing of object files ${ }^{1}$. The object-file editor provides facilities to replace, add, delete, or correct an object module. Object modules that are added to the object file are automatically reformatted into maximum-sized records to reduce both the loading time and storage requirements. In addition, the object-file editor can generate and edit DIR-type loader library files.

Many of these same facilities are also provided by the MTS linkage editor described in this volume. However, since the linkage editor makes more general assumptions concerning the reformatting and optimization process, it generally is less efficient in providing these services. The object-file editor particularly is more efficient when replacing a single module in a file containing many modules.

The object-file editor is available in the file *OBJUTIL and is invoked via the \$RUN command. The object-file editor uses the following MTS logical I/O units:

SCARDS - either the input file containing the object modules to be replaced or a sequence of commands.
SPRINT - printed output produced by the object-file editor.
SERCOM - diagnostic messages.
GUSER - user responses in conversational mode.
0 - default unit for the object file to be edited.
Those users who want only to update their object-module file can simply issue the following command:
\$RUN *OBJUTIL SCARDS=inFDname 0=editFDname
In the above case, object modules are read from "inFDname", converted to the optimized format, and written on "editFDname" replacing any previous definitions. The logical structure of the input modules is completely preserved with the following exceptions:
(1) The duplicate modules from "inFDname" are discarded.
(2) REP records of input modules are absorbed into the text of object modules.
(3) Nonabsolute DEF records are incorporated into the external symbol dictionary of the appropriate output modules.
(4) Records other than SYM, ESD, TXT, CSI, RLD, END, REP, and
${ }^{1}$ An object file normally consists of several object modules representing a main program and its subroutines.
nonabsolute DEF, are placed in front of the output modules. An example is a NCA record.

A typical example is a FORTRAN program consisting of several subroutines whose object modules are in the file OBJ. While debugging the program, the user discovers an error in one of the subroutines. This error may be corrected in the source file by the MTS file editor. After the program is recompiled, the object-file editor may be used to replace the erroneous module in the object file. The last two stages would be as follows:

```
$RUN *FTN SCARDS=PROG(200,299) SPRINT=*PRINT*
$RUN *OBJUTIL SCARDS=-LOAD 0=OBJ
```

In this example, the source code for the subroutine is in the file PROG at the specified line range. The FORTRAN compiler places the new object module into the file -LOAD which is read by the object-file editor and replaces the old version of the subroutine in the file OBJ.

The PAR field allows the user some control over the processing done by the object-file editor. Those users who wish to use the more advanced features of the object-file editor must use the object-file editor command language described below. Commands are read from SCARDS and printed output is written on SPRINT; thus, the typical \$RUN command to use the object-file editor in command mode is:
\$RUN *OBJUTIL
Command input is terminated by an end-of-file, by a RETURN command, or by a STOP command.

The following parameters may be specified in the PAR field of the \$RUN command. The parameters must be separated by commas or blanks. Parameters may be negated by "-", "ᄀ", "NO", or "N". The minimum acceptable abbreviation for each parameter is underlined.

## $B R E A K=n$

Each new module added to the edit file begins at the next highest multiple of the line number "n". Lines currently in the edit file are not changed to reflect the new value. The default value is 1.000. The BREAK parameter has no effect for sequential edit files.

COMSAVE
The COMSAVE parameter specifies that COM (comment) records are to be saved during processing. NOCOMSAVE specifies that COM records are not to be saved. The default is COMSAVE.

The DEFSAVE parameter specifies that nonabsolute DEF records are to be preserved during the object-module optimization process. NODEFSAVE specifies that nonabsolute DEF records are to be incorporated into the external symbol dictionary of the appropriate object modules. The default is NODEFSAVE.

## DIRECTORY

The DIRECTORY parameter specifies that the object-file editor is to produce a two-record DIR directory of the object modules in the edit file for future use by the object-file editor. The DIR record and its directory are written at line numbers -1 and 0 , respectively, in the edit file when the object-file editor terminates processing. This record will normally not be read by the system loader since it is in the negative line number range of the file. On subsequent uses of the object-file editor, the program will read this DIR record directory to build its internal representation of the edit file instead of reading the entire file as it must do if no DIR record exists. The presence of this DIR record will increase the file space required for the edit file but will greatly reduce the expense of replacing object modules in an edit file containing an object program consisting of a very large number of modules. The default is NODIRECTORY, i. e., the object-file editor will not produce a DIR record directory. However, if the edit file contains a DIR record at lines -1 and 0 , then DIRECTORY is assumed. The DIRECTORY parameter has no effect for sequential files.

DLR
If the DLR (delete library records) parameter is specified, library (LIB and DIR) control records will be deleted from the edit file. NODLR specifies that library records are not to be deleted. The default is NODLR.

DMD
If the $\operatorname{DMD}$ (delete multiple definitions) parameter is specified, object modules which are multiple definitions will be deleted from the edit file. NODMD specifies that multiple definitions are not to be deleted. The default is NODMD.

## EMPTY

The EMPTY parameter specifies that the edit file assigned to unit 0 is to be emptied before use. NOEMPTY specifies that the edit file is not to be emptied. The default is NOEMPTY. EMPTY has no effect in command mode.

Two hexadecimal digits "xx" specify a character to fill the gaps in the text of the control sections. The default is FILL=81.

GAPSIZE=n
"n" specifies the maximum object-module text gap size to be filled during the object-file editor processing. The default is ORL/2 or 256 , whichever is smaller.

## LIBRARY

The LIBRARY parameter specifies that the edit file (if empty) is to be formatted as a loader library file. In this case, the object-file editor will generate a DIR loader record at the beginning of the file and subsequently update the directory as object modules are added to the library. The default is NOLIBRARY, i.e., the file is not formatted as a loader library file. However, if the edit file is not empty, the object-file editor will examine the file and automatically determine whether or not it is a library file and act accordingly. Note that the LIBRARY parameter is distinct from the DIRECTORY parameter which requests the object-file editor to generate and maintain a special directory in the negative line-number range of the object file for its own use when editing.

## MISCSAVE

The MISCSAVE parameter specifies that miscellaneous (ALI, RIP, DEF, OPT, NCA, and LCS) loader records are to be preserved during processing. NOMISCSAVE specifies that miscellaneous records are to be discarded. The default is MISCSAVE.

MSGSAVE
The MSGSAVE parameter specifies that MSG (message) records are to be saved during processing. NOMSGSAVE specifies that MSG records are not saved. The default is MSGSAVE.

## OPTIMIZE

The OPTIMIZE parameter specifies that object modules are to be optimized, i.e., reformatted according to ORL, FILL, and GAPSIZE. NOOPTIMIZE specifies that input records of object modules are to be copied as is. The default is OPTIMIZE.

ORL=n
"n" specifies the maximum output record length to be used for output produced by the object-file editor. The default is the maximum output record length of the edit file (normally 32767).

QUIT
The QUIT parameter specifies that if the object-file editor encounters any errors in batch mode, the user is signed off. NOQUIT specifies that the batch user is not to be signed off if an error occurs. The default is NOQUIT.

REPSAVE
The REPSAVE parameter specifies that REP (replace) records are to be preserved during the object-module optimization process. NOREPSAVE specifies that REP records are to be absorbed into the control-section text of the appropriate object modules. The default is NOREPSAVE.

## SLOTS=n

"n" specifies the number of slots (one slot per entry point) to be allocated for the DIR record of a library file. "n" may be between 1 and 2730; the default is 128 for sequential files or the minimum (the least possible number of slots) for line files. The SLOTS parameter should be specified only if the edit file is empty and is to be formatted as a library file. Alternatively, the LIBRARY parameter may be specified if the empty edit file is to be a library file and, if a sequential file, it requires no more than 128 slots.

## SYMSAVE

The SYMSAVE parameter specifies that SYM (symbol) records are to be saved during processing. NOSYMSAVE specifies that SYM records are not saved. The default is SYMSAVE.

TERSE/VERBOSE
The TERSE/VERBOSE parameters control the amount of information produced by the verification of some commands. TERSE specifies that minimal information is requested; VERBOSE specifies that full information is desired. TERSE is an antonym for VERBOSE. The default is the setting of the MTS TERSE option. This parameter pair has no effect if NOVERIFY is specified.

## UPDATE/REPLACE

The UPDATE/REPLACE parameters control whether the file assigned to SCARDS is an update or replacement file in no-command mode. UPDATE specifies that object modules read from SCARDS not in the edit file will be included. REPLACE specifies that new modules will be excluded. The default is UPDATE.

VERIFY
The VERIFY parameter specifies that verification for each command is requested. NOVERIFY suppresses the verification. The default is VERIFY.

The complete description of the object-file editor command language is given on the following pages.

OBJECT-FILE EDITOR COMMAND LANGUAGE

The general form of a command is:
commandname[@modifier]... [ operand]...
Modifiers may be prefixed by "ᄀ", "-", "NO", or "N" if they are to be negated. In some commands, "slist" as an operand of a command stands for:
\{ALLBUT|[ONLY]\} symbol [[,]symbol] ...
The following notation conventions are used in the description of the object-file editor command language:
... denotes zero or more repetitions of the preceding words.
| denotes a choice of options, e.g., x|y means choose "x" or "y".
[ ] denotes optional words.
\{ \} denotes alternatives.
denotes a minimum acceptable abbreviation for a word, e.g., INCLUDE indicates $I$ is the minimum acceptable abbreviation for INCLUDE.

The following rules apply to command usage:
(1) A command starts with the first nonblank character, which need not start at position 1.
(2) There should be no embedded blanks in the command name and command modifiers.
(3) At least one blank should separate the command name and the first operand.
(4) There must be a blank or a comma between any two operands.
(5) If the last character of the current input line is a minus sign "-", the next input line will be taken as a continuation of the current line. The first character of the next line replaces the continuation character. There is no limit to the number of continuation lines, however, the total number of characters in a command line may not exceed 256. Note: For batch use, the continuation character must be punched in column 80, since all 80 columns of a card are read.
(6) Any command or command modifier may be abbreviated by entering only an initial substring, which is underscored in each command or command modifier description.
(7) A command line beginning with an asterisk "*" is considered to be a comment and is not processed, other than possibly being echoed by the object-file editor.
(8) A command line beginning with a dollar sign "\$" is assumed to be an MTS command and is executed by a call to the system CMD subroutine.

The following table summarizes the object-file editor commands and their applicable modifiers.

| Command | Operand | Applicable Modifiers |
| :---: | :---: | :---: |
| $\underline{\text { ADD }}$ | [FROM] FDname [slist] | BREAK=, COMSAVE, DEFSAVE, <br> GAPSIZE=, MISCSAVE, MSGSAVE, OPT, <br> ORL=, REPSAVE, SYMSAVE, TERSE, VERBOSE, VERIFY |
| CLEAR | None | VERIFY |
| CLOSE | None | None |
| COMMENT | comment | None |
| CREATE | filename [1hs=rhs] | LIBRARY, SLOTS=, VERIFY |
| CSECT | \{section\|entry\} | VERIFY |
| DELETE | slist | DIRECTORY, TERSE, VERBOSE, VERIFY |
| DISPLAY | object | LENGTH=, TYPE= |
| EDIT | filename | DLR, DMD, EMPTY, LIBRARY, SLOTS= |
| EMPTY | filename [\{OK\|O.K.|! \}] | LIBRARY, SLOTS=, VERIFY |
| EXPLAIN | [ON FDname] [item] | EMPTY |
| HELP | None | None |
| INCLUDE | [FROM] FDname [slist] | BREAK=, COMSAVE, DEFSAVE, GAPSIZE=, MISCSAVE, MSGSAVE, OPT, ORL=, REPSAVE, SYMSAVE, TERSE, VERBOSE, VERIFY |
| LIST | [ON FDname] [object [ [FOR] s | ```slist]] EMPTY, TERSE, VERBOSE``` |
| MAP | [ON FDname] [slist] | EMPTY, FULL, SORT= |
| MCMD | MTS command | None |
| MODIFY | object value | COMGEN, LENGTH=, REPGEN, TYPE=, VERIFY |
| MTS | [optional command] | None |
| PATCH | module-name | VERIFY |
| PUNCH | [ON] FDname [slist] | COMSAVE, DEFSAVE, EMPTY, GAPSIZE=, MISCSAVE, MSGSAVE, OPT, ORL=, REPSAVE, SYMSAVE, TERSE, VERBOSE, VERIFY |
| RENAME | old1[=] new1 [ [, ] old2 [=] new2] ] |  |
| REPLACE | [FROM] FDname [slist] | $\begin{aligned} & \mathrm{OM}=, ~ V E R I F Y \\ & \text { COMSAVE, DEFSAVE, GAPSIZE=, } \\ & \text { MISCSAVE, MSGSAVE, OPT, ORL=, } \\ & \text { REPSAVE, SYMSAVE, TERSE, VERBOSE, } \end{aligned}$ |

MTS 5: System Services


| Command: | $\underline{\text { ADD [ }}$ [ROM] FDname [slist] |
| :---: | :---: |
| Modifiers: | BREAK, COMSAVE, DEFSAVE, GAPSIZE, MISCSAVE, MSGSAVE, OPT, ORL, REPSAVE, SYMSAVE, TERSE, VERBOSE, VERIFY |
| Example: | ADD FILE1+FILE2 $(100,199)$ ALLBUT QQ |
| Explanation: | Modules are added to the edit file from the specified file or devices. Input is terminated by an end-of-file or by the first LDT record encountered. |
| Command: | CLEAR |
| Modifiers: | VERIFY |
| Example: | CLEAR@NV |
| Explanation: | The CLEAR command completely clears out the current internal representation of the edited file. The objectfile editor will then rebuild its internal representation by reading the edit file. If the edit file contains a DIR record, it will be reconstructed to ensure its validity. This command is necessary whenever the user changes the edit file without letting the object-file editor know (via MTS, MCMD or $\$$ commands; in particular, via an MTS \$RENUMBER command). |

Command: CLOSE
Modifiers: None
Example: CLOSE
Explanation: The CLOSE command causes the directory in the library file to be updated. Normally the directory is not updated (rewritten) until the program terminates or before a call to MTS or MTSCMD. The CLOSE command has no effect if the edit file is not a library or does not contain a special directory (via the DIRECTORY parameter).

| Command: | COMIMENT comment |
| :---: | :---: |
| Modifiers: | None |
| Example: | COMIMENT - Now we delete some symbols. |
| Explanation: | The COMMENT command is useful for documenting sequences of commands given to the object-file editor. Note that command lines beginning with an asterisk "*" are also treated as comments. |
| Command: | CREATE filename [lhs=rhs]. |
| Modifiers: | LIBRARY, SLOTS, VERIFY |
| Example: | CREATE PROG.O SIZE=100P |
| Explanaton: | The CREATE command creates an edit file. The filename is acquired and becomes the active edit file. The optional keywords are: |
|  | $\begin{aligned} & \text { SIZE=nP } \\ & \text { MAXSIZE=nP } \\ & \text { TYPE=\{LINE \|SEQ }\} \\ & \text { VOLUME=volname } \end{aligned}$ |
|  | The keywords are the same as those for the MTS command \$CREATE except that the SIZE and MAXSIZE are expressed in terms of pages. |
|  | If the LIBRARY modifier is specified, a library file is created. In this case, a DIR loader record is generated at the beginning of the file before any modules are added. The SLOTS modifier should be specified for a sequential file if the library is to contain more than 128 entry points (the default). The number of slots must be between 1 and 2730. If SLOTS is specified, the LIBRARY modifier is assumed. |
|  | Although sequential files may be created and edited, is recommended that a line file be used since deletion and replacement operations are much more efficient. |

May 1983

| Command: | CSECT \{section\|entry\} |
| :---: | :---: |
| Modifiers: | VERIFY |
| Example: | CSECT MAIN |
| Explanation: | The control section specified by "section" or containing the entry point "entry" is made the active section. This command should be given before the first DISPLAY, MODIFY, or SCAN command. |
| Command: | DELETE slist |
| Modifiers: | DIRECTORY, TERSE, VERBOSE, VERIFY |
| Example: | DELETE SUBRA, SUBRB |
| Explanation: | Through the DELETE command, the modules implied by the symbols in "slist" are deleted from the edited file. This allows the replacement of module definitions. |
|  | The DIRECTORY modifier may be specified to remove only the specified entry points from the DIR record directory, if the edit file is a loader library file. If a module has all of its entry points deleted from the directory, the module is deleted from the edit file. |
|  | The DELETE command may be used to delete modules from either line or sequential-type edit files. However, since a sequential file must be rewritten from the point of change to effect the deletion, it could be substantially more expensive to delete modules from a sequential edit file than a line file. For this reason, line files are preferred for editing. |
| Command: | DISPLAY object |
| Modifiers: | LENGTH=, TYPE= |
| Operand |  |
| Modifiers: | LENGTH=, TYPE= |
| Example: | DIS@T=I 0...20 INLOOP+4 0AC@T=XL32 |
| Explanation: | The DISPLAY command allows object-module text to be displayed in a manner similar to the equivalent debug mode command. "object" is either a single address parameter (e.g., 100) or a block address parameter (e.g., 100...1F0). An address parameter may be either a relative hexadecimal address or a symbol with an optional |

signed hexadecimal displacement (e.g., INLOOP+4). Only
symbols defined in the external symbol dictionary (i.e.,
from an ESD card) are recognized; SYM records are not
interpreted. A hexadecimal address must begin with a
decimal digit (0-9) only if it is also the name of a
defined symbol, e.g., AC is first interpreted as a symbol
name. If no symbol of that name is defined, then it is
considered as a hexadecimal number. However, 0AC is
always treated as a hexadecimal address. Each address
parameter must specify a valid relative address within
the currently active control section, as specified via
the CSECT command.
Each "object" is converted according to the type and
length specified and printed along with a one-character
code that indicates the parameter type. The type codes
are defined with the description of the TYPE modifier.

| Command: | EDIT filename |
| :--- | :--- |
| Modifiers: | DLR, DMD, EMPTY, LIBRARY, SLOTS |
| Example: | EDIT PROG. 0 |

May 1983

| Command: | EMPTY filename [\{OK\|O.K.|! \}] |
| :---: | :---: |
| Modifiers: | LIBRARY, SLOTS, VERIFY |
| Example: | EMPTY FILE.O |
| Explanation: | The specified filename is emptied and then becomes the current edit file. If the user is at a terminal, confirmation is requested before a permanent file is emptied. Confirmation is not requested for temporary files. The command may be confirmed by the response "OK", "O.K.", or "!". The confirmation may be given as the second parameter of the command. |
| Note: | This command is the same as the EDIT command with the @EMPTY modifier specified. |
| Command: | EXPLAIN [ON FDname] [item]. |
| Modifiers: | EMPTY |
| Example: | EXPLAIN @DMD, COMMANDS |
| Explanation: | The EXPLAIN command prints an explanation of the specified items on SPRINT or on FDname if specified. The following items can be explained: <br> \$, *, ?, ADD, CLEAR, CLOSE, commands, COMMENT, continuations, CREATE, CSECT, DELETE, description, DIRECTORY, DISPLAY, EDIT, EMPTY, everything, example, EXPLAIN, FILL, HELP, INCLUDE, LIST, MAP, MCMD, modifiers, MODIFY, MTS, objutil, parameters, PATCH, PUNCH, QUIT, RENAME, REPLACE, RETURN, SCAN, SET, slist, SNIFF, STOP, syntax, UPDATE, XREF, @BREAK, @COMGEN, @COMSAVE, @DEF, @DEFSAVE, @DLR, @DMD, @EMPTY, @FULL, @GAPSIZE, @LENGTH, @LIBRARY, @MISCSAVE, @MSGSAVE, @OPT, @OPTIMIZE, @ORL, @REPGEN, @REPSAVE, @SLOTS, @SYMSAVE, @TERSE, @TYPE, @UNDEF, @VERBOSE, @VERIFY. |
|  | Items listed in lowercase are generic and produce general explanation; items listed in uppercase refer to a particular command, modifier, or parameter and produce a specific explanation. The EMPTY modifier may be specified to empty the output file first. |


| Command: | HELP |
| :---: | :---: |
| Modifiers: | None |
| Example: | HELP |
| Explanation: | The HELP command prints a list of all valid object-file editor commands. |
| Command: | INCLUDE [FROM] FDname [slist] |
| Modifiers: | BREAK, COMSAVE, DEFSAVE, GAPSIZE, MISCSAVE, MSGSAVE, OPT, ORL, REPSAVE, SYMSAVE, TERSE, VERBOSE, VERIFY |
| Example: | INCLUDE FILE1+FILE2 $(100,199)$ ALLBUT QQ |
| Explanation: | Modules are added to the edit file from the specified file or devices. Input is terminated by an end-of-file or by the first LDT record encountered. |
| Note: | This is a synonym of the ADD command. |
| Command: | LIST [ON FDname] [object [[FOR] slist]] |
| Modifiers: | EMPTY, TERSE, VERBOSE |
| Example: | LIST ON *PRINT* |
| Explanation: | The LIST command allows the user to obtain information about the object modules currently in the edit file. The items of information about the modules specified by "slist" are specified by "object", where "object" may be any one of the following items: |
|  | OMS - list all module names with their corresponding beginning line numbers or sequential pointers. If "slist" is given, then the ending line number and number of lines of each specified module is also listed. |
|  | ENTRYS - list all defined symbols (types SD, LD, |
|  | LDS - list all type LD entry point symbols. |
|  | LRS - list all type LR entry point symbols. |
|  | CSECTS - list all control-section names. |
|  | SDS - list all control-section names (same as |
|  | PCS - list modules containing private control |
|  | PRS $\quad$ - list all pseudoregister definitions. ERS $\quad-\quad$ list all external references. |


| WXS | list all weak external references. |
| :---: | :---: |
| COMMONS | list all common definitions. |
| CMS | - list all common definitions (same as above). |
| GENINFO | list END record generation information. |
| MISC | - list all miscellaneous records (COM, LCS, RIP, etc.). |
| RLDS | - list the relocation dictionary (RLD) for the specified modules. The list format is similar to that of the program *OBJLIST. |
| ENDJUNK | list records at the end of the edit file. |
| COMMENTS | - list all COM (comment) records. |
| REPS | list all REP (replace) records. |
| PATCHES | list all REP records (same as above) |
| MSGS | - list all MSG (message) records. |
| OPTS | list all OPT (options) records. |
| ALIS | list all ALI (alias) records. |
| LCS | - list all LCS (low-core symbol) records. |
| LDTS | list all LDT (load terminate) records. |
| RIPS | - list all RIP (reference if present) records. |
| DEFS | - list all DEF (define) records. |
| NCAS | - list all NCA (no care) records. |
| ENTS | - list all ENT (entry) records. |
| MDLS | - list all MDL (multiple definition load- | ing) records.

DUPLICATES - list all multiply-defined entry points. URSYM - list all unreferenced symbols, i.e., all defined symbols not referenced by any type ER (external reference) symbols.
OBJSCAN - list an edited account of the contents of each specified object module. The list format is similar to that of the program *OBJSCAN.
REFERENCES - list the names of all object modules containing an external symbol dictionary (ESD) reference to the specified symbols. At least one symbol must be specified in the "slist"; the ALLBUT modifier is not allowed. This option is similar to the XREF command except that only selected symbols are cross-referenced.

If "slist" is not specified, the "object" information is listed for all relevant modules. If "object" is omitted, OM type information is printed for all object modules. If "slist" is specified, "object" or "FOR" must also be specified. LIST information is written to SPRINT unless "ON FDname" is given, in which case it is written to the specified file or device. The EMPTY modifier may be specified to empty the output file before printing the list.

| Command : | MAP [ON FDname] [slist] |
| :---: | :---: |
| Modifiers: | EMPTY, FULL, SORT= |
| Example: | MAP OBJSCAN |
| Explanation: | The external symbol dictionaries of all modules specified are printed out in symbolic form. If the FULL modifier is not specified, only control sections, label definitions, and common definitions are printed. If the FULL modifier is given, all information, including external references and pseudoregisters, is printed. Unless the SORT=NAME modifer is specified, the map output for each object module is given by order of occurrence of the symbols on the ESD records. The MAP listing is printed on the SPRINT output unless "ON FDname" is given. The EMPTY modifier may be specified to empty the output file before printing the map. |
| Command: | MCMD MTS command |
| Modifiers: | None |
| Example: | MCMD EMPTY OBJECT OK |
| Explanation: | The MTS command specified is executed by MTS, and control is returned to the object-file editor. Alternatively, the user may issue the MTS command directly in objectfile editor command mode by prefixing it with a dollar sign, e.g., \$EMPTY -OBJ. |
| Note: | The object-file editor will not know if the user changes the file being edited. |
| Command: | MODIFY object value |
| Modifiers: | COMGEN, LENGTH=, REPGEN, TYPE=, VERIFY |
| Operand |  |
| Modifiers: | LENGTH=, TYPE= |
| Example: | MOD MAIN I'STM ECDOOC,LR CF' |
| Explanation: | The MODIFY command allows object-module text to be altered in a manner similar to the equivalent debug mode command. The first parameter specifies the locations that are to be modified and the second parameter specifies the values to be used for the modification. |
|  | "object" is a single address parameter, i.e., either a valid relative hexadecimal address or a symbol with |

optional signed hexadecimal offset, which specifies the location to be modified.
"value" specifies a list of one or more constants delimited by commas and enclosed in primes. A duplication factor and/or type code may optionally prefix the "value". The duplication factor must be an unsigned decimal integer, and the type code may be any of the code types listed in the description of the TYPE modifier. Thus, the modification values may be given as an assemblerlike constant list (e.g., 18F'0', CL8'SCARDS', E'1.0, 2.0,3.0', etc.). No modifiers may be appended to "value" itself. The total length of the input constant "value" may not exceed 256 bytes.

The REPGEN modifier may be specified to cause the MODIFY command to generate REP (replace) records to effect the text modification instead of actually changing the appropriate TXT or CSI record (the default). The object-file editor will prompt the user for an optional comment to be inserted in the comment field of the REP card as follows:

Enter comment for REP card:
?
If no comment field is desired, the user may enter a null line in response to the prompt. The prompt may be suppressed by specifying the NOCOMGEN modifier and the REP card generated will have a null comment field. By default, if REPGEN is specified, COMGEN is assumed. These modifiers may also be set globally via the SET REPGEN=ON and SET COMGEN=ON commands. Currently, the REPGEN and COMGEN modifiers are only legal for edit files that are line files.

Verification of the modification is given by printing both the old value and the new value of the location modified. Verification may be suppressed by appending the modifier NV to the command, or by globally disabling verification via the SET VERIFY=OFF command.

| Command: | MTS [MTS command] |
| :--- | :--- |
| Modifiers: | None |
| Example: | MTS EMPTY OBJECT OK  <br>  MTS |
| Explanation: | If the MTS command is specified, it is executed by MTS, <br> and the object-file editor may be reentered with an MTS <br>  <br> \$RESTART command. Alternatively, the user may issue just <br> the MTS command. Control then reverts to MTS command |
|  |  |

mode in such a way that the object-file editor may be reentered with a \$RESTART command.

Note: The object-file editor will not know if the user changes the file being edited.

Command: PATCH module-name
Modifiers: VERIFY
Example: PATCH SUBR
Explanation: The PATCH command allows object-module text to be patched by inserting REP (replace) records in the object module specified by "module-name". The object-file editor will prompt the user for the patches as follows:

Enter address text <comment> :
?
Each input line will generate one REP card. Input is terminated by a null line or end-of-file. If the user enters an input line containing only the single character "D", then the previously entered patch record will be deleted. If @VERIFY if specified (the default), the object-file editor will print each REP card to be inserted and prompt the user for confirmation before inserting the REP cards into the appropriate module.

Patches may be entered in free format, i.e., leading zeros need not be supplied. However, an even number of hexadecimal digits must be specified in the text field. Commas may be used to delimit the text for readability, but only on byte boundaries. The comment field is optional but if given, it must be separated from the text field by one or more blanks.

Note that the ESDID field of the REP card is not specified by the user. The object-file editor will determine this from the patch address. If the patch address is not unique within the module, then the containing control-section name must be specified on the PATCH command to generate the proper ESDID.

The LIST command
LIST REPS [[FOR] slist]
may be used to list the REP records in a specific module or in the entire file if no "slist" is given. Currently, it is only possible to PATCH modules in a line file.

| Command: | PUNCH [ON] FDname [slist] |
| :---: | :---: |
| Modifiers: | COMSAVE, DEFSAVE, EMPTY, GAPSIZE, MISCSAV, MSGSAVE, OPT, ORL, REPSAVE, SYMSAVE, TERSE, VERBOSE, VERIFY |
| Example: | PUNCH ON - X ONLY SORT |
| Explanation: | If no "slist" is given, all modules of the current edit file are punched on FDname. Otherwise, the specified modules are punched. In addition, if the EMPTY modifier is specified, the FDname is first emptied before punching. |
| Command: | RENAME old1[=]new1 [[,]old2 [=] new2] |
| Modifiers: | OM=, VERIFY |
| Example: | RENAME OBJECT=OBJUTIL |
| Explanation: | The RENAME command causes one or more specified symbols to be renamed. Renaming is done for every occurrence of each specified symbol; all references as well as all definitions are renamed, including the symbols on ALI, DEF, ENT, LCS, NCA, and RIP records. Symbols on SYM records are not renamed. The RENAME command is useful for correcting misspelled names or for creating names that reveal content better than the symbols originally chosen. Equal signs may be omitted. <br> The renaming of a symbol may be restricted to a single object module by specifying the OM=name modifier, where "name" is any defined symbol in the desired object module. In this case, only those occurrences of the specified symbol within object module "name" will be renamed. <br> A private control section may be given a nonblank name by specifying its assigned internal name (of the form "PC\#n") as the "old" symbol. The external symbol dictionary (ESD) type of the symbol will be changed from PC (private control) to $S D$ (section definition). |
| Command: | REPLACE [FROM] FDname [slist] |
| Modifiers: | COMSAVE, DEFSAVE, GAPSIZE, MSGSAVE, OPT, ORL, REPSAVE, SYMSAVE, TERSE, VERBOSE, VERIFY |
| Example: | REPLACE FROM -LOAD |
| Explanation: | The REPLACE command reads potential replacement modules from "FDname" and selectively replaces those modules in |

the edit file according to "slist". Any additional modules in "FDname" are ignored. The REPLACE command can be thought of as a convenient way of performing the following operations:

DELETE slist INCLUDE FROM FDname slist
with the additional feature that the original ordering of the modules in the edit file is preserved if possible. Note that the UPDATE command performs a very similar function.


Command: $\quad$ SET lhs=rhs[[,]lhs=rhs]...
Modifiers: None
Example: SET COMSAVE=ON SYMSAVE=OFF
Explanation: Most of the items which can be specified in a SET command are also available as modifiers to the individual commands. The SET command simply changes the global default value for such modifiers so that the same modifier values need not be given repeatedly. The available keywords are:

| BREAK=n | Defaults to 1.000 |
| :---: | :---: |
| COMGEN $=$ \{ ON $\mid$ OFF $\}$ | Defaults to setting of REPGEN |
| COMSAVE $=\{$ ON $\mid$ OFF $\}$ | Defaults to ON |
| DEFSAVE $=\{\mathrm{ON} \mid O F F\}$ | Defaults to OFF |
| DIRECTORY= $\{$ ON $\mid$ OFF $\}$ | Defaults to OFF |
| DLR $=\{\mathrm{ON} \mid \mathrm{OFF}\}$ | Defaults to OFF |
| $\underline{D M D}=\{\mathrm{ON} \mid \mathrm{OFF}\}$ | Defaults to OFF |
| $\underline{E C H O}=\{\mathrm{ON} \mid \mathrm{OFF}\}$ | (see below) |
| ENDJUNK=name | Defaults to ENDJUNK |
| FILL=xx | Defaults to 81 |
| GAPSIZE=nnn | Defaults to min(ORL/2,256) |
| LENGTH=n | Defaults to 4 |
| LIBRARY= ON\| OFF $\}$ | Defaults to OFF |
| MODCHAR=character | Defaults to @ |
| MISCSAVE= $\left.{ }^{\text {ON }} \mid O F F\right\}$ | Defaults to ON |
| MSGSAVE $=$ \{ON $\mid$ OFF $\}$ | Defaults to ON |
| OPTIMIZE= ON $^{\text {O }}$ OFF $\}$ | Defaults to ON |
| ORL=nnn | Defaults to device maximum |
| QUIT $=\{$ ON $\mid$ OFF $\}$ | (see below) |
| REPGEN $=\{$ ON $\mid O F F\}$ | Defaults to OFF |
| REPSAVE= ON $\mid$ OFF $\}$ | Defaults to OFF |
| SLOTS $=$ n | Defaults to 128 for sequential |
|  | files, minimum for line files |
| SYMSAVE $=\{$ ON $\mid$ OFF $\}$ | Defaults to ON |
| TERSE $=\{$ ON $\mid$ OFF $\}$ | Defaults to the setting of the corresponding MTS TERSE option |
| TYPE=code | Default to X |
| VERBOSE= \{ ON\|OFF $\}$ | Defaults to the setting of the corresponding MTS VERBOSE option |

```
                                    (antonym of TERSE)
VERIFY={ON |OFF}
Defaults to ON
Note that most of these items may be specified as
execution parameters in the PAR field of the $RUN command
(see the description of the parameters available at the
beginning of this section).
If ECHO is turned ON, object-file editor commands are
ECHOed on SPRINT. ECHO defaults to ON unless the
commands are being entered directly from a terminal.
If the object-file editor encounters any errors when QUIT
is turned ON in batch mode, the user is signed off. QUIT
defaults OFF for batch and is always OFF for conversa-
tional use.
The special loader records (LCS, RIP, LDT, etc.) at the end of the edit file are considered as part of a special object module named ENDJUNK. This name will appear in the object-file editor command verification and can be used to specify this special module. If the name ENDJUNK conflicts with another object module, it may be changed via the SET ENDJUNK=name command, where "name" is from 1 to 8 characters.
```

| Command: | SNIFF |
| :--- | :--- |
| Modifiers: | None |
| Example: | SNIFF |

Explanation: This command lists the filename, its file type, the number of modules, the number of entries, and the number of object records. If the edit file is a sequential library file, the number of free slots is listed.

Command: STOP

| Modifiers: | None |
| :--- | :--- |
| Example: | STOP |

Explanation: The object-file editor terminates processing. An end-offile in the command stream also terminates the processing.

| Command: | UPDATE [FROM] FDname [slist] |
| :--- | :--- |
| Modifiers: |  |
|  | BREAK, COMSAVE, DEFSAVE, GAPSIZE, MSGSAVE, OPT, ORL, |
|  | REPSAVE, SYMSAVE, TERSE, VERBOSE, VERIFY |
| Example: |  |
|  |  |
|  | UPDATE FROM LOAD |

Command Modifiers

The modifiers are prefixed by "@" or MODCHAR as set by the SET MODCHAR command (see the SET command) and appended to the commands. A modifier may be negated by prefixing it with "ᄀ", "-", "NO", or "N".

| Modifier: | BREAK=n |
| :---: | :---: |
| Example: | ADD@BREAK=100 OBJFILE |
| Explanation: | The BREAK modifier may be appended to the ADD, INCLUDE, or UPDATE commands to set the beginning line number of each new module added to the edit file to the next highest multiple of "n". The default is 1.000. This modifier has no effect for sequential files. |
| Modifier: | COMGEN |
| Example: | MODIFY@REPGEN@っCOMGEN 240 C'WXYZ' |
| Explanation: | The COMGEN modifier may be applied to the MODIFY command to control the prompt issued by the object-file editor for the comment field to be appended to the REP card generated by the REPGEN modifier. By default, if REPGEN is specified, then COMGEN is assumed. NOCOMGEN suppresses the prompt and the REP card generated will have a null comment field. COMGEN only has effect if the REPGEN modifier is specified. |
| Modifier: | COMSAVE |
| Example: | ADD@COM MYFILE |
| Explanation: | The COMSAVE modifier causes COM records to be preserved in ADD, INCLUDE, PUNCH, REPLACE, and UPDATE commands. By default, COM records are preserved. |
| Modifier: | DEF |
| Example: | XREF@DEF |
| Explanation: | The DEF modifier may be applied to the XREF command to request the object-file editor to include in the crossreference only those external references of a module that have a definition within the edit file. This modifier is useful for determining which modules "call" which subroutines in a large, multiple-module edit file. |

May 1983

| Modifier: | DEFSAVE |
| :---: | :---: |
| Example: | INC@DEFSAVE - LOAD |
| Explanation: | The DEFSAVE modifier may be applied to the ADD, INCLUDE, PUNCH, REPLACE, and UPDATE commands to request the object-file editor to preserve nonabsolute DEF records during the object-module optimization process. By default, these DEF records are incorporated into the external symbol dictionary of the appropriate object modules, corresponding to @ DEFSAVE. |
| Modifier: | DIRECTORY |
| Example: | DELETE@DIR ONLY ENTB |
| Explanation: | The DIRECTORY modifier may be appended to the DELETE command to specify that the operands of the command are to be removed only from the DIR record directory of a loader library, but not from the library file itself. |
| Modifier: | DLR |
| Example: | EDIT@DLR PGM.O |
| Explanation: | The DLR modifier may be applied to the EDIT command to request that the object-file editor delete any library (LIB or DIR) control records in the file to be edited. The default is @ $\quad$ DLR. |
| Modifier: | DMD |
| Example: | EDIT@DMD PGM.O |
| Explanation: | The DMD modifier may be applied to the EDIT command to request the object-file editor to delete all object modules which are multiple definitions from the file to be edited. The default is @ ${ }^{\text {DMD. }}$ |
| Modifier: | EMPTY |
| Example: | PUNCH@EMPTY ON SORTFILE ONLY SORT |
| Explanation: | The EMPTY modifier may be applied to the EDIT, EXPLAIN, LIST, MAP, PUNCH, and XREF commands to request that the output file is to be emptied before output from the object-file editor is written to it. |


| Modifier: | FULL |
| :---: | :---: |
| Example: | MAP@FULL |
| Explanation: | The FULL modifier may be applied to the MAP command specify that the entire external symbol dictionary is be printed. |
| Modifier: | GAPSIZE=n |
| Example: | ADD@GAPSIZE=400 SEQFILE |
| Explanation: | When executing commands such as ADD, INCLUDE, PUNCH, REPLACE, and UPDATE, the object-file editor will fill holes of size <= GAPSIZE (for which no text was received) with a fill character, which defaults to $\mathrm{X}^{\prime \prime} 81$. . The fill character may be user-specified in the SET command with two hexadecimal digits, e.g., SET FILL=00. Each hole of size > GAPSIZE forces generation of a new TXT/CSI output record. By filling in small gaps, the number of output records can often be greatly reduced. GAPSIZE defaults to ORL/2 or 256 , whichever is smaller. |
| Modifier: | LENGTH=n |
| Example: | DISPLAY@LEN=16 MAIN+10 |
| Explanation: | The LENGTH modifier may be applied to the DISPLAY, MODIFY, and SCAN commands (or command operands) to set the length attribute to "n", where "n" is an unsigned decimal integer. The default is 4. |
| Modifier: | LIBRARY |
| Example: | CREATE@LIB PROGLIB |
| Explanation: | The LIBRARY modifier may be appended to the CREATE, EDIT, or EMPTY commands to request that a loader library be generated in the empty edit file. The default is @ LIIBRARY. This modifier has no effect if the edit file is not empty; in this case, the object-file editor automatically will determine whether or not the file is a loader library. |

May 1983

| Modifier: | MISCSAVE |
| :---: | :---: |
| Example: | ADD@-MISC *LIBRARY ONLY DTB, BTD |
| Explanation: | The MISCSAVE modifier causes miscellaneous (ALI, DEF, LCS, NCA, OPT, and RIP) records to be preserved in ADD, INCLUDE, PUNCH, REPLACE, and UPDATE commands. By default, miscellaneous records are preserved. |
| Modifier: | MSGSAVE |
| Example: | ADD@-MSG *USERS |
| Explanation: | The MSGSAVE modifier causes MSG (message) records to be preserved in ADD, INCLUDE, PUNCH, REPLACE, and UPDATE commands. By default, MSG records are preserved. |
| Modifier: | OPTIMIZE |
| Example: | REPLACE@ ${ }^{\text {OPT }}$ FILE ALLBUT IHENTRY |
| Explanation: | Object modules are normally optimized according to ORL, GAPSIZE, and FILL. No optimization forces the objectfile editor to copy object modules with no conversion. By default, the object modules are optimized. Object modules are optimized if any of the options FILL, GAPSIZE, or ORL are specified even if NOOPTIMIZE is specified. |
| Modifier: | $\underline{O R L=}$ |
| Example: | PUNCH@ORL=80 LINEFILE ONLY LAND |
| Explanation: | ORL stands for output record length and defines the maximum record size the object-file editor will write for ADD, INCLUDE, PUNCH, REPLACE, and UPDATE commands. The default for ORL is the file or device maximum. |
| Modifier: | REPGEN |
| Example: | MODIFY@REPGEN 97B4 E'3.14159' |
| Explanation: | The REPGEN modifier may be applied to the MODIFY command to request the object-file editor to generate REP (replace) records to effect the text modification instead of actually changing the appropriate TXT or CSI record. This allows a record of the text change to be kept as a separate line in the edit file. If COMGEN is in effect (defaults to ON if REPGEN is specified), the object-file |

editor will prompt for an optional comment to be inserted into the comment field of the generated REP record. The default is @ REPGEN, i.e., do not generate REP records. Currently, the REPGEN modifier is only legal for edit files that are line files.

| Modifier: | REPSAVE |
| :---: | :---: |
| Example: | PUNCH@REPSAVE TEMPOBJ |
| Explanation: | The REPSAVE modifier may be applied to the ADD, INCLUDE, PUNCH, REPLACE, and UPDATE commands to request the object-file editor to preserve REP records during the object-module optimization process. By default, REP records are absorbed into the control-section text of the appropriate object modules, corresponding to @ REPSAVE. |
| Modifier: | $\underline{S L O T S}=\mathrm{n}$ |
| Example: | EMPTY@SLOTS=200 SEQLIB |
| Explanation: | The SLOTS modifier may be appended to the CREATE, EDIT, and EMPTY commands to specify the number of slots (one slot per entry point) to be allocated for the DIR directory record of the loader library. The edit file must be empty if the EDIT command is given. The default is 128 for sequential files, or the minimum (the least possible number of slots) for line files only if the LIBRARY modifier or parameter is specified. |
| Modifier: | $\underline{\text { SORT }=\underline{N A M E}}$ |
| Example: | MAP@SORT=NAME MAIN |
| Explanation: | The SORT modifier permits sorting of MAP command output by name instead of by order of occurrence of the symbols on the ESD records. |
| Modifier: | SYMSAVE |
| Example: | INCLUDE@SYM MYFILE |
| Explanation: | The SYMSAVE modifier causes SYM records to be preserved in ADD, INCLUDE, PUNCH, REPLACE, and UPDATE commands. By default, SYM records are preserved. |

May 1983

| Modifier: | TERSE |
| :--- | :--- |
| Example: | REPLACE@TERSE FROM LOAD |
| Explanation: | The TERSE modifier may be applied to some commands to |
| abbreviate the information produced for verification. |  |
| VERBOSE is an antonym of TERSE. The default is the |  |
| setting of the MTS TERSE option. |  |


| Modifier: | UNDEF |
| :---: | :---: |
| Example: | XREF@UNDEF |
| Explanation: | The UNDEF modifier may be applied to the XREF command to request the object-file editor to include in the crossreference only those external references of a module that do not have a definition within the edit file. This action is the opposite of the DEF modifier. |
| Modifier: | VERBOSE |
| Example: | REPLACE@VERBOSE FROM -LOAD |
| Explanation: | If TERSE has been turned on globally via SET TERSE=ON, then full information for the verification of a particular command can be produced via the VERBOSE modifier. VERBOSE is the antonym of TERSE. The default is the setting of the MTS VERBOSE option. This modifier has no effect if verification is suppressed via the @ $\quad$ VERIFY modifier or the command SET VERIFY=OFF. |
| Modifier: | VERIFY |
| Example: | ADD@V -LOAD |
| Explanation: | If verification has been turned off globally via SET VERIFY=OFF, then it can be enabled for a particular command via the VERIFY modifier. |

## OBJECT-FILE EDITOR EXAMPLE

A sample terminal run is given below to illustrate several of the features of the object-file editor. In this example, user input is in lowercase while object-file editor output is in uppercase. Note also that the object-file editor uses an asterisk as the prompting character.
\#\$run *objutil
\#EXECUTION BEGINS
OBJUTIL VERSION(EP237) 12:53:25 09-27-77
*comment - example \#1 replaces two modules in an object file.
*edit gom.o
*replace from -load REPLACED:
SCOPE DSCAN
*sniff

Line file "GOM.O" has 71 modules, 182 entry points and 2277 lines.
*list oms for scope, dscan
SCOPE 12914.000 DSCAN 15330.000
**
*comment example \#2 extracts a module from a library.
**
*create sort.o
File "SORT.O" has been created.
*add from *library only sort
ADDED:
*** WARNING: Input terminated by an LDT card.
SORT SORT1 SORTEA
*map
SYMBOL TYPE ESID ADDRESS LENGTH AF
Module: SORT Size = 002E78
SORT SD $0001000000002 E 78$
SORT1 LD 0000000001
SORTEA LD 0029F0 0001
**
*comment: example \#3 is a different way to do the same thing
*comment: as in example \#2.
*edit *library
*** WARNING: The edit file "*LIBRARY" cannot be used for output
*sniff
Sequential library file "*LIBRARY" has 54 modules, 141 entry
points and 115 free slots.
*list om sort
SORT 0007004E
*punch@empty on sort.o only sort
PUNCHED:

```
MTS 5: System Services
```

```
    SORT SORT1 SORTEA
**comment example #4 creates and generates a library file.
*create@library sublib size=20p
    File "SUBLIB" has been created.
*sniff
    *** There are no object modules in "SUBLIB"
*add prog.o allbut main
    ADDED:
        READIN OUTPUT PASS1 PASS2
*include sort.o+*lcs
        INCLUDED:
            SORT SORT1 SORTEA ENDJUNK
*list
        READIN 3.000 OUTPUT 10.000
        PASS1 14.000 PASS2 18.000
        SORT 22.000 ENDJUNK 43.000
*list endjunk
        ENDJUNK - LCS LCSYMBOL
*sniff
        Line library file "-SUBLIB" has 5 modules, 7 entry points and
        43 lines.
*
*comment example #5 demonstrates editing sequential files
*edit *time
    *** WARNING: The edit file "*TIME" cannot be used for output.
*sniff
    Sequential file "*TIME" has 1 module, 1 entry point and 6
    lines.
*add *users
    *** ERROR: The edit file "*TIME" cannot be used for output.
*xref
    Module - External references
    TIME - FREESPAC GDINFO SERCOM SPRINT
    Symbol - Referenced by modules
    FREESPAC- TIME
    GDINFO - TIME
    SERCOM - TIME
    SPRINT - TIME
*stop
    CPU time = 1.69 seconds.
#EXECUTION TERMINATED 13:03:55 T=1.711 $.95
```

ACCOUNTING

## INTRODUCTION

The ACCOUNTING command allows users to display status information about their individual signon IDs. This includes such information as the amounts of spent and remaining funds, the permanent disk space allocation, and the plotting time allocations. The ACCOUNTING command, when used in this fashion, replaces the function of the *STATUS program.

The ACCOUNTING MANAGEMENT system allows project directors and instructors to distribute resources such as money, permanent disk space, and plotting time, as they wish, to various signon iDs belonging to their project or class. In addition, other constraints such as the expiration date, the maximum number of concurrent signons, and the passwords for individual signon IDs may be changed. These resource distributions are limited by maximums set for the entire project or class rather than having each signon ID with its own relatively fixed maximums. In addition, the amounts used and the maximums for any given signon ID may be displayed, as well as the amounts allocated compared with the maximums for the project and the totals for the project as a whole.

DISPLAYING STATUS INFORMATION ABOUT AN INDIVIDUAL SIGNON ID

Any signon ID may display accounting status information about its own account. The ACCOUNTING command is given in the form
\$ACCOUNTING [statusopts]
where "statusopts" are one or more options that specify either the type of information to be displayed or the format in which the information is to be displayed. The default "statusopt" in batch mode is FULL while the default in terminal mode is NOFULL.

The sample output below illustrates the format of the information presented for the FULL format.

| Status of S000 at 01/26/83 21:22 | Used | Maximum | Remaining |
| :---: | :---: | :---: | :---: |
| Cumulative charge (\$) | 125.29 | 500.00 | 374.71 |
| Permanent disk space (pages) | 51 | 100 | 49 |
| Current signons | 1 | 2 | 1 |
| Cumulative connect time (h:m:s) | 4:36:47 |  |  |
| Cumulative disk storage (pg-days) | 874 |  |  |
| Cumulative virtual memory |  |  |  |
| CPU (m:s) | 14:06 |  |  |
| Wait (pg-d h:m:s) 9d | 10:44:25 |  |  |
| Cumulative CPU time (seconds) | 15.175 |  |  |
| Magnetic tape mounts | 7 |  |  |
| Magnetic tape drive (h:m:s) | 1:21:54 |  |  |
| Cumulative page printer lines | 1275 |  |  |
| Cumulative page printer images | 250 |  |  |
| Cumulative page printer sheets | 82 |  |  |
| Cumulative line printer lines | 308 |  |  |
| Cumulative line printer pages | 10 |  |  |
| Cumulative cards read | 16 |  |  |
| Batch sessions | 1 |  |  |
| Terminal sessions | 23 |  |  |
| Expiration date and time: 12/31/83 | 24:00:00 | EDT |  | presented for the NOFULL format.


| Status of S000 at 01/26/83 21:22 | Used | Maximum | Remaining |  |
| :--- | :--- | ---: | ---: | ---: |
| Cumulative charge | (\$) | 125.29 | 500.00 | 374.71 |
| Permanent disk space | (pages) | 51 | 1 | 100 |
| Current signons |  | 1 | 2 | 49 |
| Cumulative connect time (h:m:s) | $4: 36: 47$ |  | 1 |  |

By default, the heading is always printed at the top of the status information summary. This heading may be suppressed by specifying the NOHEADING option, e.g.,

## \$ACCOUNTING NOHEADING

Individual resource options may be specified to display the remaining amounts of resources such as funds, permanent disk space, and plotter time. For example,

## \$ACCOUNTING DISK

will display the remaining (unused) number of disk pages available to the signon ID in the format

```
Remaining disk space (pages)
49
```

The individual resource options that may be specified are as follows:

```
CHARGE - Display remaining funds
DISK - Display remaining disk space
EXPIRE - Display expiration date
NETWORK - Display remaining outbound Merit network time
PLOTTER - Display remaining plotter time
SIGNONS - Display remaining number of concurrent signons
TERMINAL - Display remaining terminal connect time
```

The FULL modifier may be appended to any of the above resource options to print the used, maximum, and remaining values of a resource, e.g.,

## \$ACCOUNTING DISK@FULL

will display the used, maximum, and remaining number of disk pages for the signon ID in the format

Status of S000 at 01/26/83 21:22 Used Maximum Remaining
$\begin{array}{llll}\text { Permanent disk space (pages) } & 5100\end{array}$
Appendix A to this section gives further information about the ACCOUNTING command including all of the alternative names that may be used for the above options and modifiers.

The status information is current at the time the ACCOUNTING command is given with the exception that tape-drive time and paper-tape punched, as well as the associated charges for these, are not included for tapes currently mounted, nor are charges included for concurrent signons using the same signon ID.

The status information printed by the ACCOUNTING command is only approximate. A user's true position is given only by the monthly billing.

## THE ACCOUNTING MANAGEMENT SYSTEM

Before a project can use the ACCOUNTING MANAGEMENT system, one of the signon IDs belonging to the project must receive authorization. There can be only one authorized signon ID per project. The authorization and setting of maximums for the project and this authorized signon ID can be accomplished by contacting the Business Office at the Computing Center (764-8000).

In the examples that follow, it is assumed that the project number is SMPL, the signon ID authorized to use ACCOUNTING is S001, and the other IDs belonging to the project are S002, S003, S004, and S005. The maximum amount of money for the project is $\$ 550$, and the maximum and used amounts of money for each signon ID are as follows:

| ID | Maximum $\$$ | Used $\$$ |
| :---: | :---: | :---: |
| S001 | 100 | 25 |
| S002 | 100 | 10 |
| S003 | 100 | 125 |
| S004 | 100 | 0 |
| S005 | 50 | 15 |

Also appearing in the examples are project and signon ID maximums and used (funds, terminal time, and plotting time) or current (disk space, expiration date, and concurrent signons) signon ID amounts for each of the other values that may be changed. The current time is 4:41 p.m. on January 25, 1983.

The examples follow each other logically; that is, the result of each example is presumed to be the starting point for the next example. With the exception of the sections "Adding Money to a Signon ID" and "Producing Headings" where the complete heading is given, the one line heading in each example is for convenience only and is not part of the actual output.

The discussion of the commands, signon ranges, and keywords assumes operation from a terminal; batch operation involves only minor differences. A batch job to produce the examples in this section appears in the section "Batch Input to Produce Examples in This Section."

To start ACCOUNTING after signing on, using the authorized signon ID for the project, enter the MTS command
\$ACCOUNTING MANAGEMENT
or (in abbreviated form)
\$AC M
after the \#-sign is printed. The response will be
Project SMPL at 01/25/83 16:41 EST
and a "\$?" prefix will be printed indicating a request for input.

THE COMMANDS

The ACCOUNTING MANAGEMENT system has its own command language. The commands that manage resources are generally given in the form
command sigrange keyword
where "command" is the command verb, "sigrange" specifies the signon IDs that are to be acted upon, and "keyword" specifies the action to be taken.

May 1983

Adding Money to a Signon ID

If $\$ 100$ is to be added to the maximum charge for $S 003$, enter ADD S003 CHARGE=100

If the value of a keyword is numeric, it may, in general, be preceded by a plus or minus sign and contain a decimal point. If it is negative, it will be treated algebraically. Negative results are set to zero.

After the $\$ 100$ has been added to S 003 , ACCOUNTING will respond with

| Signon Charge | Disk | Expire Concur Term Time Plot Time |
| :---: | :---: | :---: | :---: | :---: | :---: |
| (Dollars) | (Pages) Date/Time Signon (Hrs:Min) | (Hrs:Min) |
| (Maximum above used or current amount. NC=no change on. DL=deleted.) |  |  |


| S003 | 200.00 | 10 | $12 / 31 / 83$ | 1 | Ignored |
| ---: | ---: | ---: | ---: | ---: | ---: |
| 125.00 | 2 | $24: 00 \mathrm{EST}$ |  | $0: 00$ | $0: 00$ |

and another "\$?" prefix. The first line after the three-line heading shows the new maximums for S 003 , and the second line shows the amount already used (\$125). If this is all that is desired, generating an end-of-file condition on input or issuing the STOP command will terminate and unload ACCOUNTING.

## Subtracting Money from a Signon ID

While ADD causes the values of the keyword parameters to be added to the current maximums, SUBTRACT causes the values of the keyword parameters to be subtracted from the appropriate current maximums. To reduce the maximum amount of money for S 003 by $\$ 10$, enter

SUBTRACT S003 CHARGE=10
The result will be

| Signon | Charge | Disk | Expire | Concur | Term Time | Plot Time |
| :--- | :--- | :--- | :--- | :---: | ---: | ---: |
| S003 | 190.00 |  | 10 | $12 / 31 / 83$ | 1 | Ignored |
|  | 125.00 | 2 | $24: 00 \mathrm{EST}$ |  | Ignored |  |
|  |  |  |  | $0: 00$ | $0: 00$ |  |

and a "\$?" prefix.

Modifying Money for a Signon ID

| replaced with the values of amount of money for 5002 to <br> MODIFY SOO2 CHARGE=15 |  |  |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| ACCOUNTING will respond with |  |  |  |  |  |  |
| Signon | Charge | Disk | Expire | Concur | Term Time | Plot Time |
| S002 | 15.00 | 2 | 2 12/31/83 | 1 | Ignored | Ignored |
|  | 10.00 |  | 0 24:00 EST |  | 2:00 | 0:00 |

Equalizing Money for a Signon ID

EQUALIZE causes the values of the keyword parameters to be added to the appropriate current used amounts (as opposed to the current maximums for $A D D$ ) and the corresponding maximums are replaced with the result. Thus, to give S003 \$25 more than has been used, enter

EQUALIZE S003 CHARGE=25
The result will be

| Signon | Charge | Disk | Expire | Concur | Term Time | Plot Time |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| S003 | 150.00 | 10 | 12/31/83 | 1 | Ignored | Ignored |
|  | 125.00 | 2 | 24:00 EST |  | 0:00 | 0:00 |

This command is useful for giving all the students in a class the same amount of money with which to do a new problem.

Expiring a Signon ID

The EXPIRE command, without any keyword parameters, is the same as EQUALIZE with the keyword parameters CHARGE=0, DISK=0, EXPIRE=03/01/00, CONCURSIG=1, TERMINAL=0, PLOT=0, and NOCHANGE=ON (see the section "The Keywords"). That is, all maximums except concurrent signons are set to the amount currently used, the signon ID is expired, and "no change" is set on. Thus,

EXPIRE S002

May 1983
would produce

| Signon | Charge | Disk |  | Expire | Concur |
| :---: | :---: | :---: | :---: | :---: | :---: |
| S002 | 10.00 |  | 0 | Term Time | Plot Time |
| NC | $10.01 / 25 / 83$ | 1 | $2: 00$ | $0: 00$ |  |
|  |  | 0 | $16: 41 \mathrm{EST}$ |  | $2: 00$ |
| $0: 00$ |  |  |  |  |  |

Any keyword parameters given with the EXPIRE command override the values of the corresponding implied keyword parameters. Therefore, EXPIRE S004 CHARGE=95
produces

| Signon | Charge | Disk | Expire Concur | Term Time Plot Time |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  |  |  |  |  |  |
| S004 | 95.00 | $0.01 / 25 / 83$ | 1 | $0: 00$ | $0: 00$ |  |
| NC | 0.00 | 0 | $16: 41 \mathrm{EST}$ |  | $0: 00$ | $0: 00$ |

Note that NOCHANGE=OFF does not override NOCHANGE=ON (see the section "Changing NOCHANGE").

Deleting a Signon ID

The irreversible DELETE command, like EXPIRE ("Expiring a Signon ID"), sets all maximums except concurrent signons and disk space to the amounts currently used, expires the signon ID, and sets "no change" on. The maximum number of concurrent signons is set to one and the maximum amount of disk space is set to zero. Unlike EXPIRE, the implied keyword parameters cannot be overridden, and any file space belonging to the signon ID ceases to be part of the cumulative file space for the project (see the section "PROJECT.") This latter feature means file space from the signon ID can be reassigned to other signon IDs belonging to the project even though the files have not actually been destroyed.

Because the effect of DELETE is such that the signon ID cannot be reinstated, the signon range is restricted to a single signon ID to protect against accidental deletion. In addition, if operation is from a terminal, verification will be requested. Any response to a verification request other than OK will cancel the DELETE command.

Therefore,
DELETE S005
from a terminal would produce the verification request
Enter "OK" to delete S005
and entering

OK
would allow S005 to be deleted:

| Signon | Charge | Disk |  | Expire | Concur | Term Time |
| ---: | :---: | :---: | :---: | :---: | :---: | :---: | Plot Time

The letters DL under S005 indicate that the signon ID has been deleted.
All keyword parameters that specify changes except NOCHANGE are invalid for DELETE.

In addition to the behavior produced by "no change" ("Changing NOCHANGE"), a deleted signon ID will not appear in a listing produced by DISPLAY ("Obtaining the Status of a Signon ID") unless the signon range is a single ID, the LIST=ALL keyword is included or SET LIST=ALL has been issued ("Controlling the Listing").

## Displaying a Signon ID

The DISPLAY command enables one to print the maximum and used amounts for a signon ID.

DISPLAY S001
would produce

| Signon | Charge | Disk | Expire | Concur | Term Time | Plot Time |
| :--- | ---: | ---: | :--- | ---: | ---: | ---: |
| S001 | 100.00 |  | 0 | $12 / 31 / 83$ | 1 | Ignored |
|  | 25.00 | 0 | $24: 00 \mathrm{EST}$ |  | $0: 00$ | $0: 00$ |

Note that these are the amounts as of the last signoff of the signon ID. (However, see the section "Miscellaneous" regarding charges for file storage.) All keyword parameters that specify changes are invalid for DISPLAY.

See also the section "PROJECT" for DISPLAY PROJECT.

Continuing with a Different Signon ID

The CONTINUE command may be used following any command that requires a signon range. Without any keyword parameters, it does the same thing

May 1983
as the previous command but uses the new signon range. It will not produce a new heading unless the HEADING keyword (see the section "Producing Headings") is explicitly used with the CONTINUE command. Thus, if

```
ADD S002 CHARGE=5 NOCHANGE=OFF
```

which produces

| Signon | Charge | Disk | Expire Concur | Term Time Plot Time |  |  |
| :--- | :---: | :---: | :---: | :---: | :---: | :---: |
| S002 | 15.00 |  | $001 / 25 / 83$ | 1 | $2: 00$ | $0: 00$ |
|  | 10.00 |  | $0.16: 41 \mathrm{EST}$ |  | $2: 00$ | $0: 00$ |

is followed by
CONTINUE S004
the result will be

| Signon | Charge | Disk |  | Expire | Concur | Term Time | Plot Time |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| S004 | 100.00 |  | 0 | 01/25/83 | 1 | 0:00 | 0:00 |
|  | 0.00 |  | 0 | 16:41 EST |  | 0:00 | 0:00 |

Any keyword parameters given with CONTINUE override the corresponding keyword parameters used with the previous command. Thus, after the above sequence of ADD and CONTINUE

CONTINUE S002 CHARGE=-3
would produce

| Signon | Charge | Disk | Expire Concur | Term Time | Plot Time |  |
| :--- | :---: | :---: | :---: | :---: | :---: | :---: |
| S002 | 12.00 |  | $001 / 25 / 83$ | 1 | $2: 00$ | $0: 00$ |
|  | 10.00 |  | 0 | $16: 41 \mathrm{EST}$ |  | $2: 00$ |

Note that NOCHANGE=OFF does not override the keyword parameter NOCHANGE= ON (see the section "Changing NOCHANGE").

## OTHER SIGNON RANGES

## Blocks

So far, the signon range has been a single signon ID. It may, however, refer to a group of signon IDs. For a specific, contiguous block of signon IDs belonging to the project, the signon range is the first signon ID followed by three dots (periods) and the last signon ID. For example,

DISPLAY S002...S004
would produce

| Signon | Charge | Disk | Expire | Concur | Term Time | Plot Time |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| S002 | 12.00 | 0 | 01/25/83 | 1 | 2:00 | 0:00 |
|  | 10.00 | 0 | 16:41 EST |  | 2:00 | 0:00 |
| S003 | 150.00 | 10 | 12/31/83 | 1 | Ignored | Ignored |
|  | 125.00 | 2 | 24:00 EST |  | 0:00 | 0:00 |
| S004 | 100.00 | 0 | 01/25/83 | 1 | 0:00 | 0:00 |
|  | 0.00 | 0 | 16:41 EST |  | 0:00 | $0: 00$ |

The incrementation of signon IDs within a block is defined as the collating sequence for the IBM $360 / 370$ system. That is, alphabetic letters come before the numbers 0 through 9. The first signon ID should be less than or equal to the last signon ID.

## ENTIRE

If all of the signon IDs for the project are desired, the ENTIRE parameter may be used for the signon range. For example,

SUBTRACT ENTIRE CHARGE=10
would produce the results

May 1983

| Signon | Charge | Disk | Expire | Concur | Term Time | Plot Time |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| S001 | 90.00 | 0 | 12/31/83 | 1 | Ignored | Ignored |
|  | 25.00 | 0 | 24:00 EST |  | 0:00 | 0:00 |
| S002 | 2.00 | 0 | 01/25/83 | 1 | 2:00 | 0:00 |
|  | 10.00 | 0 | 16:41 EST |  | 2:00 | 0:00 |
| S003 | 140.00 | 10 | 12/31/83 | 1 | Ignored | Ignored |
|  | 125.00 | 2 | 24:00 EST |  | 0:00 | 0:00 |
| S004 | 90.00 | 0 | 01/25/83 | 1 | $0: 00$ | 0:00 |
|  | 0.00 | 0 | 16:41 EST |  | 0:00 | 0:00 |

Note that 5005 , which has been deleted ("Deleting a Signon ID"), is not included.

Signon-Range Groups

Any combination of signon ranges except PROJECT may be grouped together by enclosing the list of signon ranges in parentheses. For example,

DISPLAY S001...S005
also could be entered as
DISPLAY (S001 S002...S004 S005)

Defined Signon-ID Ranges

A user-defined name, called a defined signon-ID range, may be given to a signon range. The name, which must begin with a + , is defined by the SET command. The range may consist of a single signon ID, a block of signon IDs, another defined signon-ID range, or a parenthesized combination of these. For exampe, the commands

SET +SINGLE=S001
SET +BLOCK=S002...S004
SET +GROUP=(+SINGLE +BLOCK)
define the signon-ID range names +SINGLE, +BLOCK, and +GROUP.
After the name +BLOCK is defined, the previous DISPLAY S002...S004 command could be given as

DISPLAY +BLOCK
Similarly, the previous SUBTRACT ENTIRE command could be given in any one of the following forms:

```
SUBTRACT (S001 S002...S004) CHARGE=10
SUBTRACT (S001 +BLOCK) CHARGE=10
SUBTRACT (+SINGLE S002...S004) CHARGE=10
SUBTRACT +GROUP CHARGE=10
```

PROJECT

The above signon ranges can be used with any command requiring a signon range except DELETE. PROJECT can be used only with DISPLAY. Entering

DISPLAY PROJECT
would produce

| Signon | Charge | Disk | Expire Concur | Term Time | Plot Time |
| :--- | :---: | :---: | :---: | :---: | :---: |
|  |  |  |  |  |  |
| PRJ | 550.00 | 50 | $12 / 31 / 83$ Ignored | Ignored | Ignored |
|  | 345.00 | $1024: 00 \mathrm{EST}$ | $2: 15$ | $0: 00$ |  |
| Tot | 337.00 |  |  |  |  |
|  | 175.00 | 2 |  | $2: 15$ | $0: 00$ |
|  |  |  |  | $0: 15$ | $0: 00$ |

No change ON for 1 and OFF for 4 of the 5 project ID's
This listing needs some explanation. The first line, labeled "PRJ" for PROJECT, shows the maximums for the project. Note the $\$ 550$ under charge. (The value under "Concur" is the maximum number of concurrent signons per signon ID.) The second line shows the cumulative amounts for the project. Since MTS permits a user to finish, once he is signed on, a signon ID maximum may be less than the corresponding used or current amount. This same condition can also be produced by using ACCOUNTING. Therefore, the cumulative amounts for the project are computed by summing the larger of the maximum and used or current amounts for each signon ID (see also the section "Miscellaneous" regarding disk space). Referring to the listings in the sections "Deleting a Signon ID" and "ENTIRE," the cumulative amount of money is computed as follows: The larger of the maximum and used amounts for S001 is $\$ 90$. The larger amount for S 002 is $\$ 10$, and the larger amounts for S003, S004, and S005 are $\$ 140$, $\$ 90$, and $\$ 15$, respectively. Adding these amounts together produces the sum of $\$ 345$.

The third line, labeled "Tot" for TOTALS, shows the sums of the maximums for the signon IDs. Thus, the sum of the maximums of $\$ 90$, $\$ 2$, $\$ 140$, $\$ 90$, and $\$ 15$ for 5001 through 5005 is $\$ 337$. If a value in this
third line is equal to the corresponding value for the project cumulative (second line), no signon IDs in the project have exceeded the signon ID maximums. If a value is less than the project cumulative value, then one or more signon IDs have used more than their allocated maximums. The fourth line shows the sums of the used or current amounts for the signon IDs. The differences between these values and the project maximums (first line) are the amounts that have not been actually used by the project; although at least some of these differences have probably been allocated to individual maximums.
(Note that the 8 pages of disk space belonging to the deleted signon ID, SOO5, are ignored in all of the above calculations.)

The fifth line states the number of signon IDs with "no change" on ("Changing NOCHANGE"), the number of signon IDs with "no change" off, and the number of signon IDs belonging to the project (5 in this case).

Remaining amounts for the project can be obtained by giving the command

DISPLAY PROJECT LIST=REMAINING
which produces

| Signon | Charge | Disk | Expire Concur | Term Time | Plot Time |
| :---: | :---: | :---: | :---: | :---: | :---: |
| PRJ | 550.00 | 50 | 12/31/83 Ignored | Ignored | Ignored |
|  | 345.00 | 10 | 24:00 EST | 2:15 | 0:00 |
| Rem | 205.00 | 40 |  |  |  |
| Div | 51.25 | 10 |  | Ignored | Ignored |
| Tot | 337.00 | 10 |  | 2:15 | 0:00 |
|  | 175.00 | 2 |  | 2:15 | 0:00 |
| Rem | 162.00 | 8 |  | 0:00 | 0:00 |

No change ON for 1 and OFF for 4 of the 5 project ID's
The line labeled "Rem" shows the results of subtracting the cumulative amounts (second line) from the maximums (first line) for the project. The line labeled "Div" shows the amounts that should be added to each signon ID with "no change" currently off in order to evenly distribute the remaining amounts. That is, the values in the "Div" line are the results of dividing the values in the "Rem" line by the number of IDs with "no change" off (\$205/4=\$51.25).

## THE KEYWORDS

## Changing Maximum Charge

We have already seen that the CHARGE keyword is used to specify changes in dollars of the maximum amount of money permitted to a signon ID. Other keywords are used to specify other maximums and control functions. All keyword parameters come after the signon range and may be listed in any order.

## Changing Maximum Disk Space

The DISK keyword is used to specify changes in pages of the maximum disk space permitted to a signon ID. To add 5 pages of disk space to S003, entering

ADD S003 DISK=5
produces

| Signon | Charge | Disk | Expire | Concur | Term Time | Plot Time |
| :--- | :--- | :--- | :--- | ---: | ---: | ---: |
| S003 | 140.00 |  | 15 | $12 / 31 / 83$ | 1 | Ignored |
|  | 125.00 | 2 | $24: 00 \mathrm{EST}$ |  | Ignored |  |
|  |  |  |  | $0: 00$ | $0: 00$ |  |

The second line of the listing under DISK is the amount of disk space (2 pages) currently being used by s003.

## Changing Expiration Time

The EXPIRE keyword is used to specify changes in the expiration time of a signon ID. The value of EXPIRE may be in either of two forms, a date form or a time form.

The date form is MM/DD/YY, where

MIM is the month,
$D D$ is the day of the month, and
YY is the year,
is used to set the expiration time to 12:00 p.m. of the specified date (leading zeros are not required). For example,

MODIFY S003 EXPIRE=6/30/83

May 1983
produces

| Signon | Charge | Disk | Expire | Concur | Term Time | Plot Time |
| :--- | :--- | :--- | :--- | ---: | ---: | ---: |
| S003 | 140.00 |  | 15 | $06 / 30 / 83$ | 1 | Ignored |
|  | 125.00 | 2 | $24: 00 \mathrm{EDT}$ |  | $0: 00$ | $0: 00$ |

The time form is MM/DD/YY@hh:mm, where
MM/DD/YY is explained above,
hh is the hour in twenty-four hour notation, and mm is the minutes,
is used to set the expiration time to a particular minute of the specified date (leading zeros and/or minutes are not required). For example,

MODIFY S004 EXPIRE=6/30/83@16:45
produces

| Signon | Charge | Disk |  | Expire | Concur | Term Time |
| :--- | :---: | :---: | :---: | :---: | :---: | :---: | Plot Time

and S004 will not be able to sign on after 4:45 p.m. on June 30, 1983.
Note that the EXPIRE keyword behaves as though it were used with MODIFY, even when used with ADD, EQUALIZE, or SUBTRACT. That is,

ADD S004 CHARGE=5 EXPIRE=6/30/83@16:45
and
SUBTRACT S004 CHARGE=5 EXPIRE=6/30/83@16:45
have the same effect on the expiration time as the above example; but the effect on the maximum amount of money is quite different in each case.

If an attempt is made to expire a signon ID before both the current time and the current expiration time, the earlier of these two times is used as the expiration time. Thus,

MODIFY S004 EXPIRE=12/31/82
would produce

| Signon | Charge | Disk |  | Expire | Concur | Term Time |
| :--- | :---: | :---: | :---: | :---: | :---: | :---: | Plot Time

when the current time is $4: 41$ p.m. on January 25, 1983.
If a signon ID is expired, there will be no charge for file (disk) space belonging to that signon ID after the expiration time. In return, the right to destroy the file space belonging to expired signon IDs is reserved by the Computing Center. However, the file space will still be considered to be part of the cumulative file space for the project until it is actually destroyed. If the expiration time of the signon ID is set to a later time before the file space is destroyed, the project may be charged for all file space used during the previously expired time.

The fact that a signon ID is expired has no effect on the changing of maximums by ACCOUNTING.

## Changing Maximum Concurrent Signons

The CONCURSIG keyword is used to specify the number of times that a signon ID may be signed on at the same time. To permit $S 003$ to have up to 3 concurrent signons, entering

MODIFY S003 CONCURSIG=3
produces

| Signon | Charge | Disk | Expire | Concur | Term Time | Plot Time |
| :--- | :--- | :--- | :--- | ---: | ---: | ---: |
| S003 | 140.00 |  | 15 | $06 / 30 / 83$ | 3 | Ignored |
|  | 125.00 | 2 | $24: 00 \mathrm{EDT}$ |  | $0: 00$ | $0: 00$ |

If the project is permitted unlimited concurrent signons per signon ID, the IGNORE parameter may be used as the value of the CONCURSIG keyword. Thus

MODIFY S003 CONCURSIG=IGNORE
produces

| Signon | Charge | Disk | Expire | Concur | Term Time | Plot Time |
| :--- | :--- | :--- | :--- | :--- | ---: | ---: |
| S003 | 140.00 |  | 15 | $06 / 30 / 83$ | Ignored | Ignored |
|  | 125.00 | $24: 00 \mathrm{EDT}$ | Ignored |  |  |  |
|  |  | $24: 00$ | $0: 00$ |  |  |  |

The word IGNORED in the listing indicates that there is no explicit limit on the number of concurrent signons.

If the value of CONCURSIG is numeric, it must be greater than or equal to 1 and less than or equal to 32767. Also, CONCURSIG always behaves as if used with MODIFY, even when used with ADD, EQUALIZE, or SUBTRACT.

## Changing Maximum Terminal Time

The TERMINAL keyword is used to specify changes to the maximum terminal time permitted to a signon ID. To restrict S 002 to 150 minutes of terminal time, entering

MODIFY S002 TERMINAL=150
produces

| Signon | Charge | Disk | Expire | Concur | Term Time | Plot Time |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| S002 | 2.00 | 0 | 01/25/83 | 1 | 2:30 | 0:00 |
|  | 10.00 | 0 | 16:41 EST |  | 2:00 | $0: 00$ |

If the project is permitted unlimited terminal time, subject only to available funds, signon IDs belonging to the project may also be permitted unlimited terminal time. Unlimited terminal time is indicated in the listing by the word IGNORED in place of the maximum. Unlimited terminal time may be set by using the IGNORE parameter as the value of the keyword TERMINAL. Thus, to give S 002 unlimited terminal time, entering

MODIFY SOO2 TERMINAL=IGNORE
produces

| Signon | Charge | Disk |  | Expire | Concur | Term Time |
| :--- | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  |  |  | Plot Time |  |  |
| S002 | 2.00 |  | $001 / 25 / 83$ | 1 | Ignored | $0: 00$ |
|  | 10.00 |  | 0 | $16: 41 \mathrm{EST}$ |  | $2: 00$ |
|  |  |  | $0: 00$ |  |  |  |

When the value is IGNORE, the behavior is the same for each of the commands ADD, EQUALIZE, EXPIRE, MODIFY, and SUBTRACT. Also, the maximum terminal time is set to zero for purposes of the line labeled TOT produced by DISPLAY PROJECT and the commands ADD and SUBTRACT.

The $H$ suffix is used to specify changes in hours of maximum terminal time permitted to a signon ID. To restrict $S 002$ to two and a quarter hours of terminal time, entering

MODIFY S002 TERMINAL=2.25H
produces

| Signon | Charge | Disk | Expire Concur | Term Time Plot Time |  |  |
| :--- | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  |  |  |  |  |  |
| S002 | 2.00 |  | 0 | $01 / 25 / 83$ | 1 | $2: 15$ |
|  | 10.00 | 0 | $16: 41 \mathrm{EST}$ |  | $2: 00$ | $0: 00$ |

Note that digits to the right of the decimal point in the value specify a decimal fraction of an hour, not minutes.

## Changing Maximum Plotting Time

The PLOT keyword is used to specify changes to the maximum plotting time permitted to a signon ID. To give S003 105 minutes of plotting time, entering

MODIFY S003 PLOT=105
produces

| Signon | Charge | Disk | Expire | Concur | Term Time | Plot Time |
| :--- | :--- | :--- | :--- | :--- | ---: | :--- |
| S003 | 140.00 |  | 15 | $06 / 30 / 83$ | Ignored | Ignored |

If the project is permitted unlimited plotting time, subject only to available funds, the word IGNORE may be used as the value of the PLOT keyword. For example,

MODIFY S003 PLOT=IGNORE
produces

| Signon | Charge | Disk | Expire Concur | Term Time Plot Time |  |
| :--- | :--- | :--- | :--- | :--- | ---: |
| S003 | 140.00 | 15 | $06 / 30 / 83$ | Ignored | Ignored |
|  | 125.00 | $24: 00 \mathrm{EDT}$ | Ignored |  |  |
|  |  | $24: 00$ | $0: 00$ |  |  |

See also the analogous discussion of unlimited terminal time in the section "Changing Maximum Terminal Time."

The $H$ suffix is used to specify changes in hours of maximum plotting time permitted to a signon ID. To give $\mathrm{SOO3}$ one hour of plotting time, entering

MODIFY S003 PLOT=1H
produces

| Signon | Charge | Disk | Expire | Concur | Term Time | Plot Time |
| :--- | :--- | ---: | ---: | ---: | ---: | ---: |
| S003 | 140.00 |  | $1506 / 30 / 83$ | Ignored | Ignored | $1: 00$ |
|  | 125.00 | 2 | $24: 00 \mathrm{EDT}$ | $0: 00$ | $0: 00$ |  |

Note that digits to the right of the decimal point in the value specify a decimal fraction of an hour, not minutes.

Filtering Signon IDs within a Signon Range

The signon IDs that are processed within a specific signon range may be restricted by the use of filters. A filter is a keyword parameter that is composed of a keyword prefixed by MAX, USED, or REM, followed by $<,<=,>,>=$, or $\neg=$, and finally a value. The keywords that may be used for filters are CHARGE, DISK, EXPIRE, CONCURSIG, TERMINAL, and PLOT. EXPIRE and CONCURSIG may only be prefixed with MAX. The values for all filters must be numeric except for MAXEXPIRE which must be a date or a time.

When filters are specified, only those signon IDs that meet the conditions specified by all of the filters will be processed. If the maximum for a signon ID is ignored, the signon ID is treated as though the maximum were infinity. Examples of filters are

```
MAXCHARGE<7.50
USEDDISK>=10
MAXEXPIRE>6/30/83
MAXCONCURSIG`=1
REMTERMINAL<=1H
USEDPLOT>30
```


## Changing NOCHANGE

When using a block or ENTIRE for the signon range with any command except DISPLAY, it may be desired that certain signon IDs within the range not be changed. If a signon ID has "no change" on, it is ignored by the command; and no listing line is produced for it unless the LIST=ALL keyword has been included or SET LIST=ALL has been entered ("Controlling the Listing"). The NOCHANGE keyword is used to specify the status of "no change". The words ON or OFF may be used as the value of the keyword. To have 5003 ignored by commands that produce changes, entering

MODIFY SOO3 NOCHANGE=ON
produces

| Signon | Charge | Disk | Expire Concur | Term Time | Plot Time |
| :---: | :---: | :---: | :---: | :---: | :---: |
| S003 | 140.00 | 15 | 06/30/83 Ignored | Ignored | 1:00 |
| NC | 125.00 | 2 | 24:00 EDT | 0:00 | 0:00 |

EQUALIZE ENTIRE DISK=2
would produce

| Signon | Charge | Disk | Expire | Concur | Term Time | Plot Time |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| S001 | 90.00 | 2 | 12/31/83 | 1 | Ignored | Ignored |
|  | 25.00 | 0 | 24:00 EST |  | 0:00 | 0:00 |
| S002 | 2.00 | 2 | 01/25/83 | 1 | 2:15 | 0:00 |
|  | 10.00 | 0 | 16:41 EST |  | 2:00 | 0:00 |
| S004 | 90.00 | 2 | 01/25/83 | 1 | 0:00 | 0:00 |
|  | 0.00 | 0 | 16:41 EST |  | 0:00 | 0:00 |

and entering
DISPLAY S003
which produces

| Signon | Charge | Disk | Expire | Concur | Term Time |
| :---: | :---: | :---: | :---: | :---: | ---: |
|  |  |  |  | Plot Time |  |
| S003 | 140.00 | $1506 / 30 / 83$ | Ignored | Ignored | $1: 00$ |
| NC | 125.00 | 2 | $24: 00 \mathrm{EDT}$ | $0: 00$ | $0: 00$ |

verifies that S 003 was not changed.
If it is now desired to add 5 dollars to S003,
ADD S003 CHARGE=5 NOCHANGE=OFF
turns off "no change" and produces
Signon Charge Disk Expire Concur Term Time Plot Time
S003 145.00 15 06/30/83 Ignored Ignored 1:00 125.00 2 24:00 EDT 0:00 0:00

To change the maximums for a signon ID that has "no change" on and leave "no change" on at the end of the operation, the keyword parameters NOCHANGE=ON and NOCHANGE=OFF should both appear. For example,

ADD S003 NOCHANGE=ON NOCHANGE=OFF DISK=2
produces

| Signon | Charge | Disk | Expire Concur | Term Time | Plot Time |
| :---: | :---: | :---: | :---: | :---: | :---: |
| S003 | 145.00 | 17 | 06/30/83 Ignored | Ignored | 1:00 |
| NC | 125.00 | 2 | 24:00 EDT | 0:00 | 0:00 |

Note that the order of the keyword parameters makes no difference.

NOCHANGE behaves the same with any command except DISPLAY, which ignores it. It can be used to inhibit the changing of a signon ID which is not currently being used. A recommended method of expiring a signon ID, for example S002, would be to first destroy all files belonging to the signon ID, and then enter

```
EQUALIZE S002 CHARGE=0 DISK=0 EXPIRE=03/01/00 CONCURSIG=1 -
TERMINAL=0 PLOT=0 NOCHANGE=ON
```

to produce

| Signon | Charge | Disk | Expire Concur | Term Time Plot Time |  |  |
| :---: | :---: | :---: | :---: | :---: | ---: | ---: |
| S002 | 10.00 |  | $001 / 25 / 83$ | 1 | $2: 00$ | $0: 00$ |
| NC | 10.00 |  | 0 | $16: 41 \mathrm{EST}$ |  | $2: 00$ |

Note that this is the same as entering
EXPIRE S002
after destroying the files.

## Changing Normal Priority Access

The NORMPRIO keyword is used to specify how the signon ID may sign on using normal priority. The four values that may be used are on, TERMINAL or $T$, BATCH or $B$, and OFF.

If the value is ON, then the signon ID will be permitted to use normal priority through either batch or a terminal. If the value is TERMINAL or $T$, then the ID will be permitted normal-priority signons from a terminal but not in batch. Likewise, if the value is BATCH or B, signons will be permitted in batch but not from a terminal. The value OFF indicates that no signons may be made using normal priority.

Therefore, if it is desired that $S 004$ not be permitted to use normal priority from batch, entering

MODIFY S004 NORMPRIO=TERMINAL
produces

| Signon | Charge | Disk | Expire | Concur | Term Time | Plot Time |
| :--- | ---: | :---: | :---: | :---: | :---: | :---: |
|  |  |  |  |  |  |  |
| S004 | 90.00 |  | $201 / 25 / 83$ | 1 | $0: 00$ | $0: 00$ |
|  | 0.00 | 0 | $16: 41 \mathrm{EST}$ |  | $0: 00$ | $0: 00$ |

The designation $N P=T$ in the listing indicates that normal priority may be used only from a terminal. No "NP=" designation would indicate that
both batch and terminal may use normal priority. NP=B and NP=OFF are used in the listing for the other normal-priority settings.

NORMPRIO produces the same behavior for each of the commands ADD, EQUALIZE, EXPIRE, MODIFY, and SUBTRACT.

Changing Low Priority Access

The LOWPRIO keyword specifies the manner in which low-priority signons may be made. The values are the same as for NORMPRIO (see the section "Changing Normal Priority Access") and have the same meaning except that they refer to low-priority usage.

MODIFY SOO4 LOWPRIO=BATCH NORMPRIO=OFF
which produces

| Signon | Charge | Disk |  | Expire | Concur | Term | Time | Plot | Time |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| S004 | 90.00 |  | 2 | 01/25/83 | 1 |  | 0:00 |  | 0:00 |
|  | 0.00 |  | 0 | 16:41 EST |  |  | 0:00 |  | $0: 00$ |
| $\mathrm{NP}=\mathrm{OFF}$ | $\mathrm{P}=\mathrm{B}$ |  |  |  |  |  |  |  |  |

## Specifying a Project Signon File

The SIGFILE keyword is used to designate a source file, or project signon file, controlled by the project director, which is invoked immediately after the user signs on and before the user's signon file is processed. That is, the behavior is the same as though a \$SOURCE command for the project signon file were issued immediately after signing on. The file may contain commands to set default local time limits, run security programs, etc.

The value for the SIGFILE keyword is a file name, OFF, or null. If the value is a file name that does not have a signon id prefix, the signon ID authorized to use ACCOUNTING is prefixed to the file name. If the file name includes a signon ID prefix, the shared file separator character following the signon ID must be a colon. If the value is OFF or null, any previous designation of a project signon file is nullified for the signon IDs in the signon range.

To designate the file INITIAL as the project signon file for S004, enter,

MODIFY S004 SIGFILE=INITIAL
which results in

| Signon | Charge | Disk | Expire Concur | Term Time Plot Time |  |  |
| :--- | ---: | :--- | :--- | :--- | :--- | :--- |
| S004 | 90.00 |  | $201 / 25 / 83$ | 1 | $0: 00$ | $0: 00$ |
|  | 0.00 | 0 | $16: 41 \mathrm{EST}$ |  | $0: 00$ | $0: 00$ |

NP=OFF LP=B
Sigfile=S001:INITIAL
The project signon file must be specified for each signon ID for which it is to be effective. Consequently, all signon IDs for the project may have the same project signon file, some may have different project signon files than others, some may not have a project signon file, etc. The project signon file designation may be changed only by using ACCOUNTING.

Since the required permit status for the project signon file depends on the contents of the file, only minimal checking can be done by ACCOUNTING. It is, therefore, the responsibility of the ACCOUNTING user to insure that the project signon file and all other files referenced directly or indirectly have the appropriate access for the signon IDs that will be using them.

SIGFILE produces the same results when used with any of the commands ADD, EQUALIZE, EXPIRE, MODIFY, or SUBTRACT.

## Inhibiting Project Signon File Attentions

The SFATTN keyword with the value OFF is used to specify that the processing of attention interrupts is to be delayed until after either the project signon file has been processed or a program run by the project signon file has set the attention interrupt exit. The value ON permits attention interrupts during the processing of the project signon file.

To inhibit attention interrupts during project signon file processing when S004 signs on, entering

MODIFY S004 SFATTN=OFF
produces

| Signon | Charge | Disk | Expire | Concur | Term Time | Plot Time |
| :--- | ---: | :--- | :--- | :--- | :--- | :--- |
|  |  |  |  |  |  |  |
| S004 | 90.00 |  | $201 / 25 / 83$ | 1 | $0: 00$ | $0: 00$ |
|  | 0.00 | 0 | $16: 21 \mathrm{EST}$ |  | $0: 00$ | $0: 00$ |

$\mathrm{NP}=\mathrm{OFF} \quad \mathrm{LP}=\mathrm{B}$
Sigfile=S001:INITIAL SA=OFF

If $S A=O F F$ does not appear in the listing, either attention interrupts are allowed during project signon file processing or it is irrelevant because there is no project signon file. For example,

MODIFY SOO4 SIGFILE=OFF
causes the listing to be

| Signon | Charge | Disk | Expire Concur | Term Time Plot Time |  |  |
| :--- | ---: | :--- | :--- | :--- | :--- | :--- |
| S004 | 90.00 |  | $201 / 25 / 83$ | 1 | $0: 00$ | $0: 00$ |
|  | 0.00 | 0 | $16: 41 \mathrm{EST}$ |  | $0: 00$ | $0: 00$ |

$\mathrm{NP}=\mathrm{OFF} \quad \mathrm{LP}=\mathrm{B}$
even though attention interrupts would still be inhibited if a new project signon file were specified.

SFATTN behaves the same with ADD, EQUALIZE, EXPIRE, and SUBTRACT as with MODIFY.

## Resetting Passwords

A password for a nonstudent signon ID may be reset by specifying the PASSWORD keyword with the MODIFY command. ACCOUNTING MANAGEMENT will prompt first for the password for the authorized ID (S001), and then for the new password for S004 followed by a confirmation of the new password. Multiple responses may be placed on the same line. For example, the command

MODIFY S004 PASSWORD
produces the prompting sequence
Enter password for s001
Enter new password for s004
Reenter new password for $\mathrm{S004}$ to confirm
The resulting output is

| S004 | 90.00 | 2 | $01 / 25 / 83$ | 1 | $0: 00$ |
| ---: | ---: | :--- | :--- | :--- | :--- |
| 0 | 0.00 | 16.41 | 000 |  |  |

$\mathrm{NP}=\mathrm{OFF} \mathrm{LP}=\mathrm{B}$
Password previously set on 02/23/82 at 16:45 EST has been changed
In batch mode, the new password must be placed on the input line immediately following the MODIFY command. A second password may be placed on the same line as the first password, in which case the two passwords must be identical.

May 1983

In order to be able to reset a password for a signon ID, three conditions must be met:
(1) the signon ID must be a nonstudent account,
(2) the signon range must be a single signon ID, and
(3) the signon ID must have previously granted permission by entering the MTS command
\$SET PROJECTPWCHANGE=ON
The setting of PROJECTPWCHANGE may be displayed by specifying the PWCHANGE keyword. For example,

DISPLAY S004 PWCHANGE
produces
S004 Pwchange=ON

Controlling the Listing

The LIST=OFF keyword parameter may be included on a command to suppress the printing of the listing line for each signon ID. For example,

ADD S001 CHARGE=15 LIST=OFF
The LIST=ALL keyword parameter causes all signon IDs in the specified signon range to be listed even though they would not normally be printed because they have "no change" on or have been deleted. For example,

MODIFY S003...S005 CHARGE=50 LIST=ALL
results in

| Signon | Charge | Disk | Expire | Concur | Term Time | Plot Time |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| S003 | 145.00 | 17 | 06/30/83 | Ignored | Ignored | 1:00 |
| NC | 125.00 | 2 | 24:00 EDT |  | 0:00 | 0:00 |
| S004 | 50.00 | 2 | 01/25/83 | 1 | 0:00 | 0:00 |
|  | 0.00 | 0 | 16:41 EST |  | 0:00 | 0:00 |
| NP=OFF LP=B |  |  |  |  |  |  |
| S005 | 15.00 | 0 | 01/25/83 | 1 | $0: 15$ | 0:00 |
| DL | 15.00 | 8 | 16:41 EST |  | 0:15 | 0:00 |

In this case $S 004$ has been modified, and S003 and S005 also appear in the listing even though they have not been changed.

The LIST=MAXIMUM and LIST=NOMAXIMUM keywords control the printing of the maximum amounts for each signon ID. The default is LIST=MAXIMUM.

The LIST=USED and LIST=NOUSED keywords control the printing of the used amounts for each signon ID. The default is LIST=USED.

The LIST=REMAINING and LIST=NOREMAINING keywords control the printing of the remaining amounts for each signon ID. The default is LIST=NOREMAINING.

An abbreviated form of the listing may be obtained by specifying the keyword parameter LIST=BRIEF. When this parameter is specified, only the values for those keywords specified with the command will be listed. In order to list values other than the ones changed, the keywords for those values may be specified without an equal sign and value. The use of a keyword in this manner implies LIST=BRIEF. If no keywords are specified and LIST=BRIEF is in effect for the DISPLAY command, values for which the maximum is not ignored are listed for charge, disk, terminal, and plot. For example,

DISPLAY SOO3 LIST=BRIEF
produces
S003 Charge=145.00,125.00 Disk=17,2 Plot=1:00,0:00
while

DISPLAY S003 EXPIRE
produces
S003 Expire=06/30/83@24:00 EDT
The default is LIST=FULL.
The defaults for each of the LIST keywords may be changed by the SET command which is described at the end of this section.

## Producing Headings

The HEADING keyword may be included with the command to produce a heading at the top of a new page before printing the next listing line. For example,

DISPLAY S001 HEADING
would produce

May 1983

| Signon | Charge (Dollars) | Disk <br> (Pages) | $\begin{gathered} \text { Expire } \\ \text { Date/Time } \end{gathered}$ | Concur <br> Signon | Term Time <br> (Hrs:Min) | Plot Time <br> (Hrs:Min) |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| (Maximu | um above u | or curr | ent amount | $\mathrm{NC}=\mathrm{no}$ | change on | DL=deleted.) |
| S001 | 105.00 |  | 2 12/31/83 | 1 | Ignored | Ignored |
|  | 25.00 |  | 0 24:00 EST |  | 0:00 | $0: 00$ |

The initial heading can be suppressed by including the NOHEADING keyword with the first command to the program that takes a signon range.

OTHER COMMANDS

The SET Command

The SET command may be used to set default parameters which are effective until the setting is changed by another SET command. The keywords that may be used with the SET command are INPUT, OUTPUT, LIST (see "Controlling the Listing"), ECHO, PROMPT, and the definitions of defined signon-ID ranges (see "Defined Signon-ID Ranges").

The INPUT and OUTPUT parameters specify the locations for input and output. To return to the default locations, enter

SET INPUT=* OUTOUT=*
The ECHO parameter controls the echoing of input in the output listing. The values for ECHO are ON (always echo), OFF (never echo), and DEFAULT or * (echo if input and output locations are different).

The PROMPT parameter specifies the character string used to prompt for input. If the string contains a blank, it must be enclosed in primes or quotation marks. PROMPT=\$? is the default.

Temporarily Returning to MTS

A temporary return to MTS may be made by entering the commands MTS or RETURN. Control may be returned from MTS to ACCOUNTING by entering a \$ACCOUNTING MANAGEMENT command. If MTS is followed by a string of characters, the string is treated as an MTS command and is executed by MTS .

Thus
MTS
or

RETURN
or
MTS CALC 100/25
leaves the session in MTS command mode. Entering
\$ACCOUNTING MANAGEMENT
produces a return to ACCOUNTING.
An input line starting with a dollar sign or MCMD indicates a command that is to be executed by MTS. An automatic return is made to \$ACCOUNTING after the command is executed. Thus, entering

MCMD CALC 100/25
or
\$CALC 100/25
would cause the CALC command to produce the answer 4, and then a "\$?" prefix would be printed indicating another input request by ACCOUNTING.

In either of the above cases, in order to save restarting costs, it is assumed that a return will be made to ACCOUNTING and that ACCOUNTING will eventually be terminated in a normal manner. Failure to terminate with an end-of-file on input or the command STOP may result in not being able to use ACCOUNTING at a later time. It should also be noted that until ACCOUNTING is terminated in this normal manner, the user is charged for approximately 40 additional pages of virtual memory.

Terminating ACCOUNTING MANAGEMENT

ACCOUNTING may be terminated and unloaded by entering the command STOP

MISCELLANEOUS

Since allocations for the signon ID that uses ACCOUNTING can be changed the same as for any other signon ID belonging to the project, care must be taken to insure that resources for this ID are not reduced too greatly. Specifically, it is quite possible to reduce the amount of
money for this signon ID to the point that once it is signed off, it will not be able to sign on again. If this happens, it is necessary to have appropriate changes made by the Computing Center Business Office (764-8000) so that the signon ID can again be used.

The cost of processing blocks, ENTIRE, and signon-range groups ("Other Signon Ranges") can be reduced by assigning the output to a printer (the default in batch or *PRINT*), a file, or *DUMMY*. The same reduction in cost can also be accomplished by using the LIST=OFF keyword (not effective for DISPLAY) with commands having a block, ENTIRE, or a group as the signon range. For example, the listing output could be assigned to *PRINT* by entering the command

```
SET OUTPUT=*PRINT*
```

The value of a maximum for a signon ID can always be reduced. However, if an increase in the value of a maximum would cause the cumulative for the project (second line produced by DISPLAY PROJECT) to exceed the maximum for the project, the increase will not be permitted. Thus, it may be possible to subtract a given amount from a signon ID and impossible to add the same amount back because the cumulative for the project originally exceeded the project maximum.

If an attempt to change the maximums for a signon ID causes an error, the maximums are not changed; no listing is produced for the signon ID; and an appropriate error comment is printed.

An attention interrupt will cause a "\$?" prefix to be printed indicating that the program is ready for the next input line. If a signon ID is being processed at the time of the interrupt, the processing, including listing of the results, will be completed before the interrupt takes effect.

The used amount of money in the listing includes charges for file (disk) storage to the current time.

It is not necessary for a project maximum to be represented completely by the corresponding signon ID maximums. That is, the project maximum may be greater than the sum of the corresponding signon ID maximums. If this is the case, to increase the maximum for a signon ID, merely enter the appropriate input line (with the provision that the new maximum will not cause the cumulative for the project to exceed the project maximum).

The project cumulative for file (disk) space is computed as described in the section "PROJECT" with one exception for compatibility with MTS. Before computing the greater of the maximum and current space for each signon ID, if the maximum is not zero, the value to be used for current space is reduced by the smaller of 16 pages and 2 plus the truncated integer result of dividing the maximum space by 8. That is, if the maximum is 0 pages, the value of the current space is used; if the maximum is 1 to 7 pages, the value to be used for current space is reduced by 2 pages; if the maximum is 8 to 15 pages, the value to be

MTS 5: System Services
used for current space is reduced by 3 pages; and if the maximum is greater than or equal to 112 pages, the value to be used for current space is reduced by 16 pages.

Both commands and keywords may be abbreviated. Only the first three letters of a command are necessary. The commands and keywords may be abbreviated as shown in the section "Accounting Management Commands" that follows.

Thus, the last example in the section "Changing NOCHANGE" can be written more concisely as

EQU S002 C=0 D=0 E=03/01/00 CS=1 T=0 P=0 NC=ON

```
Initialization File
    An accounting initialization file may be specified by the MTS command
    $SET INITFILE (ACCOUNTING)=filename
When ACCOUNTING is being initialized, a check will be made to determine
if an initialization file has been specified. If so, ACCOUNTING will
read initializing commands from the specified file before reading
commands from *SOURCE*. The initialization file facility may be
disabled by the MTS command
    $SET INITFILE (ACCOUNTING) =OFF
```


## BATCH INPUT TO PRODUCE EXAMPLES

```
$SIGNON S001 'PROJECT DIRECTOR'
password
$ACCOUNTING MANAGEMENT
ADD S003 CHARGE=100
SUBTRACT S003 CHARGE=10
MODIFY S002 CHARGE=15
EQUALIZE S003 CHARGE=25
EXPIRE S002
EXPIRE S004 CHARGE=95
DELETE S005
DISPLAY S001
ADD S002 CHARGE=5 NOCHANGE=OFF
CONTINUE S004
CONTINUE S002 CHARGE=-3
STATUS S002...S004
SUBTRACT ENTIRE CHARGE=10
DISPLAY PROJECT
DISPLAY PROJECT LIST=REMAINING
ADD S003 DISK=5
MODIFY S003 EXPIRE=6/30/83
MODIFY S004 EXPIRE=6/30/83@18:00
MODIFY S004 EXPIRE=12/31/82
MODIFY S003 CONCURSIG=3
MODIFY S003 CONCURSIG=IGNORE
MODIFY S002 TERMINAL=150
MODIFY SOO2 TERMINAL=IGNORE
MODIFY S002 TERMINAL=2.25H
MODIFY S003 PLOT=105
MODIFY S003 PLOT=IGNORE
MODIFY S003 PLOT=1H
MODIFY SOO3 NOCHANGE=ON
EQUALIZE ENTIRE DISK=2
DISPLAY S003
ADD S003 CHARGE=5 NOCHANGE=OFF
ADD S003 NOCHANGE=ON NOCHANGE=OFF DISK=2
EQUALIZE S002 C=0 D=0 E=03/01/00 CS=1 T=0 P=0 NC=ON
MODIFY SO04 NORMPRIO=TERMINAL
MODIFY SOO4 LOWPRIO=BATCH NORMPRIO=OFF
MODIFY S004 SIGFILE=INITIAL
MODIFY S004 SFATTN=OFF
MODIFY S004 SIGFILE=OFF
MODIFY S004 PASSWORD
newpassword
ADD S001 CHARGE=15 LIST=OFF
MODIFY S003...S005 CHARGE=50 LIST=ALL LIST=ALL
DISPLAY S003 LIST=BRIEF
DISPLAY S003 EXPIRE
DISPLAY SOO1 HEADING
STOP
```

ACCOUNTING MANAGEMENT COMMANDS

The following notation conventions are used in the prototypes of the commands:
lowercase - represents a generic type which is to be replaced by an item supplied by the user.
uppercase - indicates material to be repeated verbatim in the command.
brackets [] - indicates that material within the brackets is optional.
braces \{\} - indicates that the material within the braces represents choices, from which exactly one must be selected. The choices are separated by vertical bars.
ellipsis ... - indicates that the preceding syntactic unit may be repeated.
underlining - indicates the minimum unambiguous form of the command or parameter. Longer abbreviations are accepted.

The following pages give a complete summary of the commands in the Accounting Management command language.

```
ADD sigrange keyword ... [filter ...]
    sigrange signon-ID
            signon-ID...signon-ID
            +groupname
                        ({signon-ID|signon-ID...signon-ID|+groupname} ...)
                        ENTIRE
    keyword CHARGE=dollars
        {CONCURSIG|CS}={number |IGNORE }
        DISK=pages
        EXPIRE=date-time
                        {LOWPRIO|LP}={ON|\underline{TERMINAL |\underline{BATCH |OFF}}}\mathbf{|}=\underline{ON}
                        {NOCHANGE |NC}={ON |OFF}
                        {NORMPRIO|NP}={ON|\underline{TERMINAL |BATCH |OFF}}\{一|
        PLOT={mm|hhH|IGNORE}
            {\underline{SATTN |SA } ={ON|OFF}}
            {SIGFILE|SF}=[{filename|OFF}]
            TERMINAL={mm|hhH|IGNORE}
            {HEADING|NOHEADING|NHEADING}
            LIST={\underline{ALL|ON|OFF}}
            LIST={\underline{F}ULL|BRIEF}
            LIST={MAXIMUM| NOMAXIMUM}
            LIST= {USED | NOUSED}
            LIST={\underline{REMAINING \NOREMAINING}}
    filter {M[AX]|U[SED]|R[EM]}keyword operator value
CONTINUE sigrange [keyword ...] [filter ...]
DELETE signon-ID [option ...]
    option {NOCHANGE |NC}={ON|OFF}
            {HEADING NOHEADING|NHEADING}
            LIST={ELLL|ON|OFF}
            LIST={EULL|BRIEF}
            LIST={\underline{MAXIMUM | NOMAXIMUM}}
            LIST={\underline{USED | NOUSED}}
            LIST={\underline{REMAINING |NOREMAINING}}
DISPLAY {sigrange|PROJECT} [keyword ...]
EQUALIZE sigrange keyword ... [filter ...]
EXPIRE sigrange [keyword ...] [filter ...]
HELP [topic]
MCMD mts-command
```

```
MTS 5: System Services
Page Revised September 1985
May 1983
MODIFY sigrange keyword ... [filter ...]
MTS [mts-command]
RETURN
SET option ...
    option ECHO={ON |OFF | DEFAULT | * }
            INPUT={FDname | * }
            LIST={\underline{ALL}|ON|OFF}
            LIST={FULL|BRIEF}
            LIST={MAXIMUM| NOMAXIMUM}
            LIST= {USED |NOUSED}
            LIST={REMAINING|NOREMAINING}
            OUTPUT={FDname|*}
            PROMPT={string|'string'|"string"}
            +groupname=sigrange
STOP
SUBTRACT sigrange keyword ... [filter ...]
```


## ADD

Accounting Command Description

Purpose: To increase the maximum resource allotments.
Prototype: ADD sigrange keyword ... [filter ...]
Action: The ADD command may be used to increase the maximum resource allotments for a signon ID or a group of signon IDs. "sigrange" specifies the signon ID or group of IDs and may be one of the following:
signon-ID
A single signon-ID may be specified, e.g., S003.
signon-ID...signon-ID

A contiguous block of signon IDs belonging to the project may be specified, e.g.,

S002...S004.
All of the signon $1 D$ s ranging from the first ID to the last ID are included in the block.
+groupname
A defined signon-ID range may be specified (see the SET command description for details).
(\{signon-ID|signon-ID...signon-ID|+groupname\} ...)
A parenthesized list of signon IDs, blocks of signon IDs, or defined signon-ID ranges may be specified, e.g.,
(S001 S002...S004 S005).
ENTIRE
The ENTIRE parameter may be specified to include all of the signon IDs in the project.

The following "keyword" parameters are used to specify the particular resource that is to be increased.

The CHARGE parameter specifies the amount of money that is to be added to the maximum charge allotment for the signon ID or IDs. The charge is given in units of dollars with decimal points and minus signs permitted, e.g., CHARGE= 50 will add $\$ 50$ to the account(s) while CHARGE= -50 will delete $\$ 50$ from the account.
\{CONCURSIG|CS $\}=\{$ number $\mid \underline{I G N O R E}\}$
The CONCURSIG parameter sets the number of allowable concurrent signons for a signon ID or IDs, e.g., CONCURSIG=3 sets the number of concurrent signons to 3 . The value must be greater than zero. The IGNORE parameter may be used to specify an unlimited number of concurrent signons.

DISK=pages
The DISK parameter specifies the number of disk pages that is to be added to the maximum disk space allotment, e.g., DISK=25 adds 5 disk pages to the current allotment. Minus signs are permitted.

EXPIRE=date
The EXPIRE parameter sets the expiration date and time for a signon ID or IDs. The date and time is given in the form "mm/dd/yy@hh:mm", e.g., EXPIRE=6/30/87@18:00 sets the expiration date and time to 6 pm of June 30 , 1987. The minute portion of the time may be omitted, in which case ":00" is assumed. The entire time portion also may be omitted, in which case Midnight is assumed, e.g., EXPIRE=12/31/87 sets the expiration date and time to Midnight of December 31, 1987.
$\{\underline{L O W P R I O} \mid \mathrm{LP}\}=\{O N|\underline{T E R M I N A L}| \underline{B A T C H} \mid \underline{O F F}\}$
The LOWPRIO parameter specifies the manner in which LOW-rate access may be made. ON allows both terminal and batch access, TERMINAL allows terminal access only, BATCH allows batch access only, and OFF allows no access during LOW-rate periods.
$\{$ NOCHANGE $\mid \mathrm{NC}\}=\{\mathrm{ON} \mid \underline{O F F}\}$
The NOCHANGE parameter sets the "no change" status of a signon ID. When NOCHANGE=ON for a signon ID, subsequent accounting commands that change values will ignore the signon ID.
$\{\underline{\text { NORMPRIO }} \mid$ NP $\}=\{O N|\underline{T} E R M I N A L| \underline{B A T C H} \mid \underline{O F F}\}$
The NORMPRIO parameter specifies the manner in which NORMAL-rate access may be made. ON allows both terminal and batch access, TERMINAL allows terminal access only, BATCH allows batch access only, and OFF allows no access during NORMAL-rate periods.
$\underline{\underline{P L O T}}=\{\mathrm{mm}|\mathrm{hhH}| \underline{\text { IGNORE }}\}$
The PLOT parameter specifies the amount of plot time to be added to the maximum plot-time allotment. "mm" specifies minutes of plot time, "hhH" specifies hours of plot time, and IGNORE specifies an unlimited amount of plot time. Decimal points and minus signs are permitted.

$$
\{\underline{S F A T T N} \mid S A\}=\{O N \mid \underline{O F F}\}
$$

The SFATTN=OFF parameter specifies that the processing of attention interrupts is to be delayed until after either the project sigfile has been processed or a program run by the project sigfile has set the attention interrupt exit. The SFATTN=ON parameter permits attention interrupts during the processing of a project sigfile.
\{SIGFILE|SF $\}=[\{$ filename|OFF $\}]$
The SIGFILE parameter specifies a project sigfile to be executed when the signon ID or IDs sign on. The project sigfile is controlled by the project director and is processed when the signon ID signs on before the ID's own sigfile is processed. SIGFILE=OFF or SIGFILE= disables the project sigfile.

TERMINAL $=\{m m|h h H|$ IGNORE $\}$
The TERMINAL parameter specifies the amount of terminal time to be added to the maximum terminal-time allotment. "mm" specifies minutes of terminal time, "hhH" specifies hours
of terminal time, and IGNORE specifies an unlimited amount of terminal time. Decimal points and minus signs are permitted.

The following "keyword" parameters control the format and extent of the output produced by the command.
\{HEADING|NOHEADING|NHEADING\}
The HEADING parameter forces a skip to the next page followed by the printing of the heading. NOHEADING suppresses the page skip and the printing of the heading. HEADING is effective only when LIST=FULL is in effect.
$\underline{L I S T}=\{\underline{A L L}|O N| \underline{O F F}\}$
LIST $=\{\underline{M} A X I M U M \mid$ NOMAXIMUM $\}$
LIST= \{USED |NOUSED\}
LIST $=$ \{REMAINING $\mid$ NOREMAINING $\}$
$\underline{L I S T}=\{\underline{F} U L L \mid \underline{B} R I E F\}$
The LIST parameter controls the type and extent of output produced by the command. LIST=ALL produces an entry in the list for each signon ID in the signon range regardless of whether it was changed. LIST=ON produces an entry for each signon that was actually changed. LIST= OFF suppresses the listing. The default is LIST=ON.

LIST=MAXIMUM produces for each entry a line giving the maximum amounts allotted to each resource. LIST=NOMAXIMUM suppresses the printing of that line. The default is LIST=MAXIMUM.

LIST=USED produces for each entry a line giving the used amounts for each resource. LIST= NOUSED suppresses the printing of that line. The default is LIST=USED.

LIST=REMAINING produces for each entry a line giving the remaining amounts for each resource. LIST=NOREMAINING suppresses the printing of that line. The default is LIST=NOREMAINING.

LIST=FULL produces the listing in column format. LIST=BRIEF produces the listing in keyword format and only prints the keywords for the resources that were actually changed. The default is LIST=FULL.

The above LIST parameters may be specified either individually or in a parenthesized list, e.g., LIST=(ALL,REMAINING).

In addition, resource keywords without values may be specified to request the printing of that resource. Such a keyword specification implies BRIEF format, e.g.,

ADD S001 DISK CHARGE=10
will print the disk and new charge values in keyword format.

## PWCHANGE

The PWCHANGE parameter may be specified to display the setting of the MTS \$SET PROJECTPWCHANGE option by a signon ID, e.g.,

ADD S001 PWCHANGE CHARGE=10
will display the setting of the PROJECTPWCHANGE option and the new charge value. PWCHANGE implies BRIEF format.
"filter" parameters may be given to specify restrictions on the "sigrange" that is processed by the command. The filter is specified in the following form:
\{M [AX]|U[SED]|R[EM]\}keyword operator value
The $M, U$, and $R$ abbreviations are not initial substring abbreviations, i.e., MCHARGE or MAXCHARGE are valid but MACHARGE is invalid.

The keywords that may be filtered are CHARGE, CONCURSIG, DISK, EXPIRE, PLOT, and TERMINAL. The operators that may be used are <, <=, >, >=, and $7=$. Only MAX may be used with CONCURSIG and EXPIRE. Examples are:

MAXCHARGE<7.50
USEDDISK>=10
MEXPIRE>6/30/83
MCS $ᄀ=10$
REMTERMINAL<=60
UPLOT>30
Ingored maximums are treated as infinity, e.g., UPLOT>30 includes IDs for which PLOT=IGNORE is in effect.

With the above examples, only the signon ID or IDs in "sigrange" that satisfy the filtering restriction are processed, e.g., the command ADD S001...S009 CHARGE=10 USEDCHARGE<100 increases the maximum charge allotment by $\$ 10$ only for those signon IDs in the sigrange that have used less than $\$ 100$.

## CONTINUE

Accounting Command Description

Purpose: To continue the action of the previous command.
Prototype: CONTINUE sigrange keyword ... [filter ...]
Action: The CONTINUE command may be used to continue the action of the previous ADD, EQUALIZE, EXPIRE, MODIFY, or SUBTRACT command except with a new "sigrange" and, optionally, new "keyword" parameters. Any new keyword parameters specified on the CONTINUE command override the same parameters given on the previous command. For example, the command sequence

ADD S001 CHARGE=10 DISK=10
CON SOO4 DISK=5
is equivalent to the command sequence
ADD S001 CHARGE=10 DISK=10
ADD S004 CHARGE=10 DISK=5

The "sigrange", "keyword", and "filter" parameters available are the same as for the ADD command.

DELETE
Accounting Command Description

| Purpose: | To delete a signon ID. |
| :---: | :---: |
| Prototype: | DELETE signon-ID [option ...] |
| Action: | The DELETE command may be used to delete a single signon |
|  | ID. The maximum resource allotments (charge, terminal |
|  | time, and plot time) are set to their current used |
|  | values, the disk space allotment is set to zero, the |
|  | number of concurrent signons is set to one, the expiration date is set to the current time, and the "no change" |
|  | flag is set. All disk space belonging to the signon ID |
|  | is returned to the project and may be reassigned to other signon IDs. |
|  | Verification is requested. Any response other than OK will cancel the command. |
|  | Only the HEADING, NOHEADING, and LIST formatting options and the NOCHANGE parameter may be specified. They are described with the $A D D$ command. |
|  | A deleted signon ID may not be reinstated. |

May 1983

DISPLAY
Accounting Command Description

Purpose: To display the status of resource allotments.
Prototype: DISPLAY \{sigrange|PROJECT\} [kwyword ...]
Action: The DISPLAY command may be used to display the current values for the used and maximum resource allotments for a signon ID, a group of signon IDs, or a project.

The "sigrange" and "keyword" parameters available are the same as for the ADD command, except that values are not permitted on keywords, e.g.,

DISPLAY DISK CHARGE

## EQUALIZE

Accounting Command Description

Purpose: To equalize the maximum resource allotments.
Prototype: EQUALIZE sigrange keyword ... [filter ...]
Action: The EQUALIZE command may be used to equalize the maximum resource allotments for a signon ID or a group of signon IDs. The equalization is performed by adding the resource values to the used (instead of the maximum) amounts to set a new maximum allotment. For example, the command

EQUALIZE ENTIRE CHARGE=25
will equalize the maximum charge allotments of the project so that each signon ID has $\$ 25.00$ available in unused funds.

The "sigrange", "keyword", and "filter" parameters available are the same as for the ADD command.

EXPIRE
Accounting Command Description

Purpose: To "expire" a signon ID or group of signon IDs.
Prototype: EXPIRE sigrange keyword ... [filter ...]
Action: The EXPIRE command may be used to "expire" a signon ID or a group of signon IDs. All of the maximum resource allotments (charge, disk space, terminal time, and plot time) are set to their current used values, the number of concurrent signons is set to one, the expiration date is set to the current time, and the "no change" flag is set.
"keyword" parameters may be specified to override the setting of the above-mentioned resource allotments. In particular, the maximum disk space may set to zero by specifying DISK=-32768 so that the disk space used by the signon ID or IDs will become automatically available to the project when the files are destroyed.

An "expired" signon ID may be reinstated with the ADD or MODIFY commands by setting NOCHANGE=OFF, setting a new expiration date, and adding funds.

The "sigrange", "keyword", and "filter" parameters available are the same as for the ADD command.

```
MTS 5: System Services
Page Revised September 1985
May 1983
                    HELP
            Accounting Command Description
Purpose: To obtain on-line assistance for ACCOUNTING.
Prototype: HELP [topic]
Action: The HELP command provides on-line assistance for the use
of ACCOUNTING. If no parameter is specified, general
assistance is provided; otherwise, assistance on a spe-
cific "topic" is provided. For further information,
enter the command
    HELP HELP
The HELP command is under the control of the MTS command
    SET HELPMODE={LINE|SCREEN}
```

Purpose: To modify the maximum resource allotments.
Prototype: MODIFY sigrange keyword ... [filter ...]

Action: The MODIFY command may be used to change the maximum resource allotments for a signon ID or a group of signon IDs. The MODIFY command is similar to the ADD command except that that the resource allotments are set to a particular value instead of being increased by a particular value.

The "sigrange", "keyword", and "filter" parameters available are the same as for the ADD command.

In addition, the following parameter may be specified by nonstudent projects for the MODIFY command:

PASSWORD
The PASSWORD parameter may be specified by a project director to change the password of a single signon ID if the owner of that ID has issued the MTS command \$SET PROJECTPWCHANGE=ON, e.g.,

MODIFY S001 PASSWORD
The project director will be prompted for both the password of the current signon ID and the new password of the signon ID being changed. In terminal mode, there will be a prompt to enter the new password a second time for verification. If desired, the responses to these prompts may be entered on the same line in which case the prompting message will be suppressed. In batch mode, the verification is not required.

MTS 5: System Services

May 1983


RETURN

| Accounting Command Description |  |
| :--- | :--- |
| Purpose: | To return to the caller. |
| Prototype: | RETURN |
| Action: | The RETURN command returns to the caller (normally MTS <br> Command mode). This does not terminate the invocation of <br> the \$ACCOUNTING system. |

Purpose: To set accounting-management system options.
Prototype: SET option ...

Action: The SET command may be used to change the status of various accounting-management system options. More than one option may be specified on the SET command; one or more blanks must separate each option.

```
ECHO={ON |OFF|DEFAULT |*} Default: DEFAULT
```

The ECHO option controls the echoing of input commands. ECHO=ON specifies that input lines are always echoed on the output device. ECHO=OFF specifies that input lines are never echoed. ECHO= DEFAULT or ECHO=* specifies that input lines are echoed only if the output device is different from the input device.

```
INPUT={FDname|*} Default: *
```

The INPUT option specifies the file or device from which input lines are read. By default, input lines are read from * which is the current setting of the MTS \$SOURCE command. If an attention interrupt is given, the input is reset to *MSOURCE*. An end-offile on input terminates the \$ACCOUNTING system.

| LIST $=\{$ ALL \|ON |OFF\} | Default: | ON |
| :--- | :--- | :--- |
| LIST $=\{$ MAXIMUM\|NOMAXIMUM\} | Default: | MAXIMUM |
| LIST $=\{$ USED\|NOUSED\} | Default: | USED |
| LIST=\{REMAINING\|NOREMAINING\} | Default: | NOREMAINING |
| LIST $=\{F U L L \mid$ BRIEF\} | Default: | FULL |

The LIST parameter controls the type and extent of output produced by the command. LIST=ALL produces an entry in the list for each signon ID in the signon range regardless of whether it was changed. LIST=ON produces an entry for each signon that was actually changed. LIST=OFF suppresses the listing.

LIST=MAXIMUM produces for each entry a line giving the maximum amounts allotted to each resource. LIST=NOMAXIMUM suppresses the printing of that line.

LIST=USED produces for each entry a line giving the used amounts for each resource. LIST=NOUSED suppresses the printing of that line.

LIST=REMAINING produces for each entry a line giving the remaining amounts for each resource. LIST= NOREMAINING suppresses the printing of that line.

LIST=FULL produces the listing in column format. LIST=BRIEF produces the listing in keyword format and only prints the keywords for the resources that were actually changed.

The above LIST parameters may be specified either individually or in a parenthesized list, e.g., LIST= (ALL, REMAINING) .

```
OUTPUT={FDname |* } Default: *
The OUTPUT option specifies the file or device to which output lines are written. By default, output lines are written to * which is the current setting of the MTS \$SINK command. If an attention interrupt is given, the output is reset to *MSINK* if the input is not being read from *MSOURCE*.
```

```
PROMPT={string|'string'|"string"}
```

Default: \$?

The PROMPT option sets the accounting-management system prompting prefix. The prompting prefix may be from zero to sixteen characters in length. If the prefix string contains blanks or commas, it must be enclosed in primes or quotes.

```
+groupname=sigrange
```

A user-defined signon-ID range may be defined. The name of the range must begin with a plus sign (+). The contents of the range may consist of a single signon ID, a block of signon IDs, another defined signon-ID range, or a parenthesized combination of these, e.g.,

SET +SINGLE=S001
SET + BLOCK=S002...S004
SET +GROUP=(S001 +BLOCK S005...S009)
The defined signon-ID range may be used with any command requiring a signon-ID range parameter, e.g.,

```
        DISPLAY +BLOCK
        EQUALIZE (+BLOCK S005...S009) CHARGE=10
```

May 1983

STOP
Accounting Command Description

Purpose: To terminate the accounting-management system session and return to the caller.

Prototype: STOP
Action: The accounting-management system session is terminated and control is returned to the caller (normally MTS command mode). All system workspace and buffers are released.

## SUBTRACT

Accounting Command Description

Purpose: To decrease the maximum resource allotments.
Prototype: SUBTRACT sigrange keyword ... [filter ...]
Action:
The SUBTRACT command may be used to decrease the maximum resource allotments for a signon ID or a group of signon IDs.

The "sigrange", "keyword", and "filter" parameters available are the same as for the ADD command.

APPENDIX A: DISPLAY STATUS INFORMATION

The ACCOUNTING command may be used to display status information for any signon ID. The ACCOUNTING command is given in the form

ACCOUNTING [statusopts]
where "statusopts" specify one or more options that may be used to selectively filter the status information given about the current signon ID. The options are as follows:

FULL
Print all information. In addition to the items listed under NOFULL below, the following quantities are printed:
amount of temporary file space cumulative figures for file storage CPU and wait-memory used
CPU time used
number of tape mounts tape-drive time used lines, images, sheets, and pages printed cards read and punched paper tape punched plotter paper used number of batch and terminal sessions expiration date and time

FULL is the default in batch mode if no options other than HEADING or NOHEADING are specified.
\{NOFULL | $\neg$ FULL |- FULL $\}$
Print the maximum, used, and remaining figures for the following items:

## charge

current file space
concurrent signons
terminal time
plotter time
external network time
NOFULL is the default in conversational mode if no options other than HEADING or NOHEADING are specified.

## HEADING

Print a heading before the next line that contains a used amount. This is the default for the first such line printed.
\{NOHEADING| ᄀHEADING|-HEADING\}
Do not print a heading. If this option is specified, it should be first.
\{ㄷHARGE | DOLLARS |FUNDS | \$ \}
Print the remaining amount of funds for the user.
\{DISK|FILE\}
Print the remaining amount of file space for the user.
EXPIRE
Print the expiration date and time for the user.
NETWORK

Print the remaining amount of outbound network connect time available to the user.

PLOTTER
Print the remaining amount of plotter time available to the user.

SIGNONS
Print the remaining number of concurrent signons permitted for the user.
\{TERMINAL|CONNECT\}
Print the remaining amount of terminal connect time available to the user.

One of the following modifiers may be appended to the CHARGE, DISK, SIGNONS, TERMINAL, PLOTTER, or NETWORK parameters or their synonyms. If a modifier is to apply to more than one parameter, the parameters may be separated by commas and grouped within parentheses, e.g., (\$,DISK)@D.
\{@DETAILED|@FULL|@NOREMAINING|@ᄀREMAINING|
@ - REMAINING\}
Print the maximum, used, and remaining figures for the modified quantities rather than only the remaining amounts.

## \{@REMAINING|@NODETAILED|@าDETAILED|@-DETAILED | @NOFULL|@ ${ }^{\text {@FULL }}$ @- FULL\}

Print only the remaining amounts for the modified quantities. This is the default if a modifier is not specified.

The ACCOUNTING command may be used to print information regarding the signon IDs charge; current and cumulative file space; signons; terminal, plotting, and network time; CPU and wait-memory use; CPU time; I/O; and expiration time.

If the command is given in conversational mode and no options are specified (other than HEADING or NOHEADING), the items listed for the parameter NOFULL are printed. If the command is given in batch mode and no options are specified (other than HEADING or NOHEADING), the items listed for the parameter FULL are printed in addition to those listed with NOFULL. If all information about an item is zero, no information normally is printed unless the item is specifically specified as a "statusopt". The information is current at the time the command is given with the exception that tape drive time and paper tape punched as well as the associated charges for these are not included for tapes currently mounted, nor are charges included for a concurrent signon using the same signon ID.

It must be emphasized that the information printed is only approximate. A user's true position is indicated only by the monthly billing.

## APPENDIX B: MULTILEVEL ACCOUNTING

In addition to being able to allocate resources to a group of signon IDs, Accounting Management may be set up to allow a higher-level manager to allocate resources to lower-level managers who in turn can reallocate the resources to the individual signon IDs. For example, a course instructor may allocate funds to teaching assistants who then may allocate those funds to the individual students.

Each multilevel accounting structure may be viewed as a tree. Each of the branches is an accounting level; each of the leaves on a branch is a signon ID.

Each accounting level will have one or more signon IDs and zero or more lower accounting levels attached to it. The one signon ID which is always attached to an accounting level is the signon ID that does the accounting management for that level.

Each accounting level controls the maximums for the signon IDs and levels attached directly to it. The maximums for both signon IDs and levels are controlled by their immediately higher level, not subsequently higher levels.

The used quantities for a signon ID, as always, are the resources that have been expended or are currently in use. The used quantities for an accounting level are the resources that have been allocated to lower levels and signon IDs, whether or not these resources have actually been expended. If the used amount for a lower level or signon ID is greater than the maximum allocated, then this used amount is included in the used amount of the higher level.

The use of Accounting Management with higher levels is the same as described for single levels with the following differences:
(1) In the listing, an $L$ beneath the signon ID indicates that this is an accounting level rather than a signon ID. The level should be specified on commands to Accounting Management by using the level ID, which is the same as the signon ID that manages the level. Accounting Management knows whether the ID should refer to an accounting level or to a signon ID.
(2) Only signon IDs have passwords and sigfiles. Therefore, the PASSWORD, PWCHANGE, SIGFILE, and SFATTN keywords are not applicable to accounting levels.
(3) The DELETE command may not be used for accounting levels.
(4) To determine how much has actually been used by all signon IDs belonging to all lower levels, use the LIST=ACTUALUSED keyword. This will print these amounts as an additional set of quantities in the listings for the accounting levels. For single levels that contain only signon IDs, this is the same information as
the amounts used by the level, but it requires additional processing time to produce it.

| Examples of Multilevel Projects

MTS 5: System Services
<EFL>, 103
<FIX>, 103
\$ACCOUNTING command, 239
*LIBRARY, 84
*LINKEDIT, 159
*OBJLIST, 87
*OBJSCAN, 87
*OBJUTIL, 97, 207
A linkage editor modifier, 180
Account status, 240
Accounting, 239
Commands, 242
File space, 245, 252, 252
Heading, 264
Keywords, 242, 250
Listing, 263
Project maximums, 239, 250
Signon ranges, 242, 248, 273
Signon-range filters, 257, 276
Signon-range groups, 249, 273, 287
Status, 246, 250
Totals, 250
ADD Accounting command, 243, 273
ADD linkage editor command, 165
ADD object-file editor command, 215
Adding, Accounting, 243
Additional parameters, SORT, 25-28
Address sort, 57, 67
ALI record, 89, 142, 198
ALIAS linkage editor command, 165
ARL SORT parameter, 29
ATTRIBUTE linkage editor command, 165
Average record length, SORT, 29, 51

Batch priority, 259, 260
BC linkage editor modifier, 181, 197

BLAST linkage editor command, 166
Block data subprogram, 94
Block of signon ID's, 248
Blocking, 11, 12, 21-24, 49, 50, 52, 77-80
Blocking utility, (See Blocking)
BREAK object-file editor modifier, 230
BREAK object-file editor parameter, 208
BTCRPL option, 92
Byte, 12
CALL option, 37, 73
Character sorts, 13-04, 18, 34, 72, 74
Charge, 239
CHARGE Accounting keyword, 243, 274
CHECK linkage editor modifier, 181
CHK SORT parameter, 26
CLEAR linkage editor command, 166
CLEAR object-file editor command, 215
CLOSE object-file editor command, 215
Collating field, 12, 18, 37, 48, 51, 72-76
Collating sequence, 12, 19, 34, 35, 37, 48, 56, 72-76
COM record, 89, 139, 198
COMBINE linkage editor command, 167, 195
COMGEN object-file editor modifier, 230
Commands, Accounting, 242
COMMENT linkage editor command, 167
COMMENT object-file editor command, 216
Common section, 93, 94
COMSAVE linkage editor modifier, 181
COMSAVE linkage editor parameter,

$$
160
$$

COMSAVE object-file editor modi-
fier, 230
COMSAVE object-file editor parameter, 208
Concurrent signons, 254
CONCURSIG Accounting keyword, 254, 274
CONTINUE Accounting command, 246, 279
Control section, 86, 93
Control statement, SORT, 17, 57, 62, 63, 65, 67, 67, 68, 81
Copy, 18
COPY linkage editor command, 168
CREATE object-file editor command, 216
Csect, 86, 93, 94
CSECT linkage editor command, 168
CSECT linkage editor modifer, 182
CSECT object-file editor command, 217
CSI record, 116, 194, 196
Current link level, 148
Current option, 149
Current storage level, 148, 158
Data sets, SORT, (See also Input and Output), 21-24, 75
Deblocking, (See Blocking)
Deblocking utility, (See Blocking)
DEBUG command, 84, 99, 147
DEC SORT parameter, 26
DEF object-file editor modifier, 230
DEF record, 89, 138, 198
Defined sequence, 20, 34, 56, 74
DEFSAVE object-file editor modifier, 231
DEFSAVE object-file editor parameter, 209
DEL SORT parameter, 26, 51, 56
DELETE Accounting command, 245, 280
DELETE linkage editor command, 168
DELETE object-file editor command, 217
Deleting a signon ID, 245
DIO SORT parameter, 29
DIR library, 98
DIR record, 97, 145, 198
DIRECTORY object-file editor modifier, 231

DIRECTORY object-file editor parameter, 209
DISK Accounting keyword, 252, 274
Disk space, 245, 252
DISPLAY Accounting command, 246, 281
DISPLAY linkage editor command, 169, 204
DISPLAY object-file editor command, 217
DLR object-file editor modifier, 231
DLR object-file editor parameter, 209
DMD object-file editor modifier, 231
DMD object-file editor parameter, 209
DPI SORT parameter, 29
DUMP linkage editor command, 170
Duplicate records, SORT, 26, 56
Dynamic loader, 83
Dynamic loading, 98, 99
ECHO Accounting option, 265, 287
EDIT object-file editor command, 218
Efficiency, SORT, 20, 24, 27, 28, 34, 42, 49, 50, 56, 58
EMPTY linkage editor modifier, 182
EMPTY linkage editor parameter, 160
EMPTY object-file editor command, 219
EMPTY object-file editor modifier, 231
EMPTY object-file editor parameter, 209
END record, 86, 104, 118
END SORT parameter, 25
ENT record, 89, 104, 138, 198
ENTIRE Accounting keyword, 248, 273
ENTRY linkage editor modifer, 183
ENTRY linkage editor modifier, 182
Entry point, 104
EQUALIZE Accounting command, 244, 282
Equalizing, Accounting, 244
ERRMAP option, 110
Errors, SORT, 29, 30, 39, 40, 51, 59, 63, 66, 68
ESD record, 86, 113, 194, 195

ESID, 87
Exits, SORT, 30-41, 41, 49, 63, 67, 68
Expiration time, 252
EXPIRE Accounting command, 244, 283
EXPIRE Accounting keyword, 252, 274
Expiring a signon ID, 244
EXPLAIN object-file editor command, 219
External dsect, 94
External symbols, merging, 94
FDUB, SORT, 31, 36, 39, 40, 57
File space, 245, 252
FILL object-file editor parameter, 210
FILL option, 91
FORTRAN output, sorting, 15-06
FREESPAC subroutine, 148, 158
FULL linkage editor modifier, 183
FULL object-file editor modifier, 232
Funds, 239
GAPSIZE linkage editor modifier, 183
GAPSIZE linkage editor parameter, 160
GAPSIZE object-file editor modifier, 232
GAPSIZE object-file editor parameter, 210
GENSAVE linkage editor modifer, 184
GENSAVE linkage editor parameter, 160
GETSPACE subroutine, 88, 147, 158

Halfword, 12
HEADING Accounting keyword, 264, 276
Heading, Accounting, 264
HELP object-file editor command, 220
Highest option, 149, 158
Highest storage level, 148, 158
I/O error, SORT, 29, 30, 39, 40, 51
INCLUDE linkage editor command, 170

INCLUDE object-file editor command, 220
Index number, 148
Initial ESD list, 100
INPUT Accounting option, 265, 287
Input, SORT, 21-24, 29, 31, 33, 39, 48, 49, 50, 51, 57, 75

Keywords, Accounting, 242, 252
LCS record, 90, 140, 198
LCSYMBOL, 103
LDT record, 90, 104, 137, 198
LENGTH linkage editor modifer, 184
LENGTH object-file editor modifier, 232
LIB record, 95, 143, 199
LIBRARY object-file editor modifier, 232
LIBRARY object-file editor parameter, 210
LIBSRCH option, 84
Line library, 98
Link level, 148
LINK subroutine, 99, 147
Linkage editor, 159
Linkage editor commands, 162
LIO SORT parameter, 27
LIST Accounting keyword, 263, 276
LIST Accounting option, 287
LIST linkage editor command, 170
LIST object-file editor command, 220
Listing, Accounting, 250, 263
LOAD command, 84, 99, 147
LOAD subroutine, 99, 147
Loader, 83
Loader map, 110
LOADINFO subroutine, 88
Low priority, 260
Low-core symbol dictionary, 101
LOWPRIO Accounting keyword, 260, 274

Magnetic tapes, SORT, 28, 50
Map, 111
MAP linkage editor command, 172
MAP object-file editor command, 222
Maximum storage index number, 148
MBY SORT parameter, 29
MCMD Accounting command, 285
MCMD linkage editor command, 172

MCMD object-file editor command, 222
MDL option, 91
MDL record, 90, 140, 199
Merge, 11, 12, 18, 20, 31
MERGE bit, 100
MISCSAVE linkage editor modifer, 184
MISCSAVE linkage editor parameter, 160
MISCSAVE object-file editor modifier, 233
MISCSAVE object-file editor parameter, 210
MNR SORT parameter, 27
MODIFY Accounting command, 244, 284
MODIFY linkage editor command, 172, 200, 204
MODIFY object-file editor command, 222
Modifying, Accounting, 244
Money, 239
MSG record, 90, 141, 199
MSGSAVE linkage editor modifier, 185
MSGSAVE linkage editor parameter, 160
MSGSAVE object-file editor modifier, 233
MSGSAVE object-file editor parameter, 210
MTS Accounting command, 285
MTS linkage editor command, 174
MTS object-file editor command, 223

NAME linkage editor modifier, 185
NCA record, 90, 139, 199
NCALCS option, 91
NOCHANGE Accounting keyword, 257, 275
Normal priority, 259
NORMPRIO Accounting keyword, 259, 275
Notation, SORT prototype, 16
NULMSG option, 91
NV linkage editor modifier, 185
Object module, 83, 85, 205
Object module library, 95
Object-File Editor, 207
OM linkage editor modifier, 185

OPT record, 90, 142, 199
Optimization, SORT, (See Efficiency)
OPTIMIZE object-file editor modifier, 233
OPTIMIZE object-file editor parameter, 210
ORL linkage editor modifier, 186
ORL linkage editor parameter, 160
ORL object-file editor modifier, 233
ORL object-file editor parameter, 210
OUTPUT Accounting option, 265, 287
Output, SORT, 21-24, 29, 35, 36, 40, 49, 50, 51, 57, 75

Page, 147
Parameters, SORT, 25-28
Parameters, SORT numeric, 58, 63, 66, 68
PASSWORD Accounting keyword, 262, 284
PATCH object-file editor command, 224
PDMAP option, 111
PGNTTRP, SORT, 30
PGS SORT parameter, 29
PLOT Accounting keyword, 256, 275
Plotting time, 256
PL1SYM, 103
POINT library, 98
POP option, 91
Predefined map, 111
Priority, 259, 260
Private control section, 94
PRMAP option, 110
Program interrupt, SORT, 29, 30, 40, 51
Program mask, SORT, 30
PROJECT Accounting keyword, 250
Project maximums, 239, 250
Project status, 250
PROMPT Accounting option, 265, 287
Psect, 94
Pseudo-register, 93
Pseudo-register map, 110
PUNCH linkage editor command, 174
PUNCH object-file editor command, 225
PURGE linkage editor command, 174, 196
PUSH option, 91

PWCHANGE Accounting keyword, 276
QUIT object-file editor parameter, 211

REC SORT parameter, 27
Record, 12
Relocation factor, 111
RENAME linkage editor command, 175
RENAME object-file editor command, 225
REP record, 92, 137, 199
REPGEN object-file editor modifier, 233
REPLACE linkage editor command, 175
REPLACE object-file editor command, 225
REPLACE object-file editor parameter, 211
REPSAVE object-file editor modifier, 234
REPSAVE object-file editor parameter, 211
RERUN command, 84, 99, 147
RES SORT parameter, 27, 50
Resetting passwords, 262
RESTART command, 149
Restricted parameters, SORT, 29
RETURN Accounting command, 286
Return codes, SORT, 30, 31, 33, 35, 37, 39, 40, 41, 57
RETURN linkage editor command, 176
RETURN object-file editor command, 226
RIP record, 92, 96, 144, 199
RLD record, 86, 117, 194, 196
RUN command, 84, 99, 147
SAVESD option, 91
SCAN linkage editor command, 176, 200
SCAN object-file editor command, 226
Section map, 111
Segment, 147
Sequential library, 97
SET Accounting command, 249, 265, 287
SET linkage editor command, 177
SET object-file editor command, 227
SETIOERR, SORT, 30

Setting global parameters, Accounting, 265
SFATTN Accounting keyword, 261, 275
SIG, SORT, 27, 31, 33, 35, 39, 40, 41
SIGFILE Accounting keyword, 260, 275
SIGFILE attentions, project, 261
SIGFILE, project, 260
Signon file attentions, project, 261
Signon file, project, 260
Signon ranges, 242, 248, 273
Signon-range filters, 257, 276
Signon-range groups, 249, 273, 287
Signons, concurrent, 254
SLI linkage editor modifier, 186, 197
SLOTS object-file editor modifier, 234
SLOTS object-file editor parameter, 211
SNIFF object-file editor command, 228
SORT, (See also item desired or Table of Contents), 11, 12, 18 Address sort, 57, 67 Average record length, 29, 51 Blocking, 12, 21-24, 49, 50, 52, 77-80
Character sorts, 13-04, 18, 34, 72, 74
Collating field, 12, 18, 37, 48, 51, 72-76
Collating sequence, 12, 19, 34, 35, 37, 48, 56, 72-76
Control statement, 17, 57, 62, 63, 65, 67, 67, 68, 81
Copy, 18
Data sets, 21-24, 75
Defined sequence, 20, 34, 56, 74
Duplicate records, 26, 56
Efficiency, 20, 24, 27, 28, 34, $42,49,50,56,58$
Errors, 29, 30, 39, 40, 51, 59, 63, 66, 68
Exits, 30-41, 41, 49, 63, 67, 68
FORTRAN output, 15-06
Input, 21-24, 29, 31, 33, 39, 48, 49, 50, 51, 57, 75

Merge, 12, 18, 20, 31
Mode, 18
Notation, prototype, 16
Numeric parameters, 58, 63, 66, 68
Output, 21-24, 29, 35, 36, 40, 49, 50, 51, 57, 75
Program interrupt, 29, 30, 40, 51
Program mask, 30
Restart, 27, 50
Return codes, 30, 31, 33, 35, 37, 39, 40, 41, 57
Simple character sorts, 13-04
Subroutine, 57, 57-70
Tape-merge facility, 28
Timing, 42-47, 60, 64, 67, 69
Virtual data set, 57, 58, 62, 65
SORT linkage editor modifier, 186
SORTEA, 41
SORTE0, 30, 30-41
SORTE1, 31, 49, 54
SORTE2, 33, 49, 51
SORTE3, 35, 49, 51, 51
SORTE4, 36, 49, 51, 54
SORTE5, 37, 49, 63, 67, 68, 73
SORTE6, 30, 39
SORTE7, 30, 40
SORTE8, 30, 40, 63, 67, 68
SORTE9, 41
SORT2, 57, 62-64
SORT3, 57, 65-67
SORT4, 57, 67-70
SPIE, SORT, 30
START command, 149
Statistics, SORT, 51
Status, Accounting, 246, 250
STOP Accounting command, 289
STOP linkage editor command, 178
STOP object-file editor command, 228
Storage index number, 100, 148
SUBTRACT Accounting command, 243, 290
Subtracting, Accounting, 243
SYM record, 86, 119
SYMSAVE linkage editor modifer, 187
SYMSAVE linkage editor parameter, 160
SYMSAVE object-file editor modifier, 234

SYMSAVE object-file editor parameter, 211
SYMTAB option, 88, 99
TERMINAL Accounting keyword, 255, 275
Terminal priority, 259, 260
Terminal time, 255
TERSE linkage editor modifer, 187
TERSE linkage editor parameter, 161
TERSE object-file editor modifier, 235
TERSE object-file editor parameter, 211
Timing, SORT, 42-47, 60, 64, 67, 69
Totals, Accounting, 250
TPS SORT parameter, 28
TXT record, 86, 115, 194, 196
TYPE linkage editor modifier, 187
TYPE object-file editor modifier, 235

Unblocking, (See Blocking)
Unblocking utility, (See Blocking)
UNDEF object-file editor modifier, 236
UNLINK linkage editor command, 178
UNLOAD command, 148
UNLOAD subroutine, 100, 148, 158
UPDATE linkage editor command, 179
UPDATE object-file editor command, 229
UPDATE object-file editor parameter, 211

V linkage editor modifier, 188
VERBOSE linkage editor modifer, 188
VERBOSE linkage editor parameter, 161
VERBOSE object-file editor modifier, 236
VERBOSE object-file editor parameter, 211
VERIFY linkage editor parameter, 161
VERIFY object-file editor modifier, 236
VERIFY object-file editor parameter, 212
Virtual data set, 57, 58, 62, 65

May 1983

Virtual memory, 147
WARN option, 91
Word, 12
WXLCS option, 92
WXTOER linkage editor modifer, 188
WXTOER linkage editor parameter,

161
XCTL subroutine, 99, 147
XREF linkage editor command, 179
XREF object-file editor command, 229

MTS 5: System Services

May 1983

System Services
Volume 5
May 1983

Errors noted in publication:

Suggestions for improvement:

Your comments will be much appreciated. The completed form may be sent to the Computing Center by Campus Mail or U.S. Mail, or dropped in the Suggestion Box at the Computing Center, NUBS, or UNYN.

## Date

Name

Address

Publications
Computing Center
University of Michigan
Ann Arbor, Michigan 48109

Volume 5
May 1983

Updates to this manual will be issued periodically as errors are noted or as changes are made to MTS. If you desire to have these updates mailed to you, please submit this form.

Updates are also available in the memo files at the Computing Center, NUBS, and UNYN; there you may obtain any updates to this volume that may have been issued before the Computing Center receives your form. Please indicate below if you desire to have the Computing Center mail to you any previously issued updates.

Name

Address $\qquad$
$\qquad$

Previous updates needed (if applicable): $\qquad$

The completed form may be sent to the Computing Center by Campus Mail or U.S. Mail, or dropped in the Suggestion Box at the Computing Center, NUBS, or UNYN. Campus Mail addresses should be given for local users.

```
Publications
Computing Center
The University of Michigan
Ann Arbor, Michigan 48109
```

Users associated with other MTS installations (except the University of British Columbia) should return this form to their respective installations. Addresses are given on the reverse side.

```
Addresses of other MTS installations:
    Publications Clerk
    352 General Services Bldg.
    Computing Services
    The University of Alberta
    Edmonton, Alberta
    Canada T6G 2H1
    Information Officer, NUMAC
    Computing Laboratory
    The University of Newcastle upon Tyne
    Newcastle upon Tyne
    England NE1 7RU
    Rensselaer Polytechnic Institute
    Documentation Librarian
    3 1 0 ~ V o o r h e e s ~ C o m p u t i n g ~ C e n t e r ~
    Troy, New York 12181
    Simon Fraser University
    Computing Centre
    User Services Information Group
    Burnaby, British Columbia
    Canada V5A 1S6
    Wayne State University
    Computing Services Center
    Academic Services Documentation Librarian
    5950 Cass Ave.
    Detroit, Michigan 48202
```


[^0]:    ${ }^{1}$ See the description of the GDINFO subroutine in MTS Volume 3, System Subroutine Descriptions.

[^1]:    ${ }^{1}$ See the description of the subroutine PLCALL in MTS Volume 7, PL/I in

