M T S

The Michigan Terminal System

Volume 2X: Public File Descriptions

April 1982

(Revised January 1987)

The University of Michigan Computing Center
Ann Arbor, Michigan

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 U-M Computing News, Computing Center Memos, and future Updates to this volume for the latest information about changes to MTS.

April 1982

PREFACE

This volume contains the public file descriptions of programs that are either obsolete or seldom used. In an effort to reduce the size the MTS Volume 2, Public File Descriptions, these descriptions were moved to this volume.

Richard A. Salisbury

General Editor

Contents

*ALGOL

Contents: The MTS version of the OS/360 ALGOL compiler.

Purpose: To compile main programs and external procedures written in the ALGOL programming language.

Use: The ALGOL compiler is invoked by the $RUN command.

Program Key: *ALGOL

Logical I/O Units Referenced:

SCARDS -	source program to be compiled (defaults to *SOURCE*).
SPRINT -	source program listing, identifier table, storage table, and diagnostic and error messages (defaults to *SINK*).
SPUNCH -	compiled object module.
SERCOM -	diagnostic and error messages (defaults to *MSINK*) (conversational mode only).

Description: This compiler is the MTS version of the IBM IBM OS/360 compiler. The definition of the language accepted by this compiler is given in the IBM publication <u>IBM System/360 Operating System ALGOL Language</u>, form GC28-6615. Information concerning the compiler and its use in OS/360 is given in <u>IBM System/360 Operating System ALGOL Programmer’s Guide</u>, form GC33-4000; a portion of this is relevant to MTS, and in particular, sections 3 and 4 and appendices C and F.

The ALGOL execution-time library *ALGOLLIB is used in conjunction with the object modules produced by the ALGOL compiler. Further information necessary to run ALGOL programs in MTS is given below and in the *ALGOLLIB public file description in this volume.

Compiler Options:

The following options may be specified in the PAR field of the $RUN command. The entries may appear in any order (except for the FIXED option) and, if any are missing, a standard default is assumed. Following each parameter in the list below is an abbreviated form for the option. The default for each option is underlined.

FIXED If this parameter is specified, the source program is assumed to be in card image form; columns 1-72 are compiled while columns 73-80 are ignored. If this option is omitted, the source program is free format up to 255 characters per line; nothing is ignored. Furthermore, all "tab" characters are converted to blanks. If this option is specified, <u>it must be given first</u>.

<u>PROGRAM</u>	(PG)	This specifies that the source program is an ALGOL main program.
PROCEDURE	(PC)	This specifies that the source program is an ALGOL external procedure.
<u>SHORT</u>	(SP)	This specifies that short-precision (4 bytes) floating-point representation is used for real values.
LONG	(LP)	This specifies that long-precision (8 bytes) floating-point representation is used for real values.
<u>LOAD</u>	(L)	This specifies that an object module is written on SPUNCH.
NOLOAD	(NL)	This specifies that no object module is produced.
<u>SOURCE</u>	(S)	This specifies that the source program and identifier table listings are printed on SPRINT.
NOSOURCE	(NS)	This specifies that these listings are not produced.
<u>EBCDIC</u>	(EB)	This specifies that the character set used for the source program is the 53-character subset of EBCDIC.
ISO	(I)	This specifies that the character set used is the 46-character BCD set established as standard for ALGOL by ISO and DIN (ISO); see the <u>IBM System/360 Operating System ALGOL Programmer's Guide</u> for details.
TEST	(T)	This specifies that the object module is to include code normally used for program testing, i.e., for checking array bounds for subscripting, and for tracing the execution of the ALGOL program. See the *ALGOLLIB description for further information.
<u>NOTEST</u>	(NT)	This specifies that the object program is not to include this code.

Restrictions:

(1) No input record from SCARDS may be longer than 255 characters.
(2) SPUNCH must be assigned to a <u>line</u> <u>file</u>. The compiler <u>always</u> writes part of the object module starting in line 1 and part starting in line 100. It is not possible to put the object module of an ALGOL compilation into a file containing the object module from a previous compilation without first emptying the file.
(3) The following temporary files are used during an ALGOL compilation; they may not be sequential files:

-ALGUT1, -ALGUT2, and -ALGUT3

(4) If used, the FIXED parameter must be the first parameter given in the parameter field.
(5) The DECK parameter must _not_ be specified in the parameter field.
(6) The SIZE=n parameter will be _ignored_ in the parameter field.

*ALGOLLIB

Contents:	The execution-time library for programs compiled by *ALGOL.
Purpose:	To provide the subroutines necessary to run ALGOL programs in MTS.
Use:	*ALGOLLIB should be concatenated to the file containing the compiled ALGOL program on the $RUN command, i.e.,

```
     $RUN object+*ALGOLLIB
```

Program Key: *EXEC

Logical I/O Units Referenced:

SCARDS	-	ALGOL program input data.
SPRINT	-	ALGOL program output data.
SERCOM	-	error messages.
2 - 9	-	ALGOL program input or output data.

Parameters: The following parameters may be specified in the PAR field of the $RUN command.

CODE=name	specifies that the file "name" contains precompiled procedures to be loaded if called by the ALGOL program. If used, this parameter _must be specified first_. The total length of "name" may not exceed 40 characters.
TRACE	specifies that the semicolon count of every statement executed is printed.
TRBEG=n	where "n" is a decimal number, specifies that the semicolon count is printed starting with semicolon "n".
TREND=n	where "n" is a decimal number, specifies that the semicolon count is printed until semicolon "n" is reached.
DUMP	specifies that if an error occurs during execution, a printout of pertinent information will occur. See the _IBM System/360 Operating System ALGOL Programmer’s Guide_, form GC33-4000, for a description of this dump.

Both TRBEG and TREND may be specified. If one of these is given and TRACE is omitted, then the trace will not extend to separately compiled procedures called within the range given, while if TRACE is specified, such procedures will also be traced. Any program to be traced must have been compiled with the TEST parameter on the $RUN *ALGOL command.

Separately compiled external procedures may be included as follows:

(1) The external procedures should be compiled individually and separately from the main program; the PROCEDURE parameter should be specified for each procedure. The outermost 'BEGIN' and 'END' statements should not appear in the source program in this case. For example:

```
$RUN *ALGOL SCARDS=PROC1 SPRINT=-P
     SPUNCH=OBJPROC1 PAR=PROCEDURE,LONG
```

where PROC1 contains, for instance:

```
'REAL' 'PROCEDURE' F(X,Y); 'REAL' X,Y;
'BEGIN' Y:=3.0; F:=(X+2.3)**4.2;
'END'
```

(2)
```
$RUN *GENLIB SCARDS=OBJPROC1+OBJPROC2+ ...
     +OBJPROCN SPUNCH=PROCLIB
```

where OBJPROC1, OBJPROC2, ..., OBJPROCN are the object modules of previously compiled procedures, as above. PROCLIB will contain the object module of the procedures, in a suitable library format.

(3)
```
$RUN OBJPROC1+*ALGOLLIB SCARDS=DATA1
     SPRINT=ANSWERS1 4=WORKFILE PAR=CODE=PROCLIB
```

It is possible to concatenate procedure libraries, for example:

```
PAR=CODE=*KDFLIB+PROCLIB
```

The correspondence between ALGOL data set numbers and MTS logical I/O units is as follows:

Data Set Number	Logical I/O Unit
0	SCARDS
1	SPRINT
2 - 9	2 - 9
10 - 14	Illegal
15	File -ALGLDD15

Data sets 0-9 cannot be positioned, i.e., the control procedure SYSACT cannot be called with a second parameter of 4 or 14 for these data sets. Data set 15 always corresponds to the temporary file -ALGLDD15 (which _must_ be a line file), and it may be positioned. This file will be emptied before use. The maximum length for SCARDS is 120 and the output record length for SPRINT is

121. The default record length for 2 through 9 and -ALGLDD15 is 80, but this can be changed by calling SYSACT with a code of 6 before the file is opened. See also the IBM publication IBM System/360 Operating System ALGOL Language, form GC28-6615.

If semicolon count tracing is specified for all or part of the program, the file -ALGUT1 will be used and it must be a line file.

If the PUT or GET predefined procedures are used, the file -ALGUT2 will be used and it also must be a line file.

Example:

```
$RUN ALGOLPROG+*ALGOLLIB SCARDS=DATA SPRINT=RESULTS
     5=-WORK
```

In the above example, the ALGOL object program in the file ALGOLPROG is executed. The input is read from the file DATA and the output is written into the file RESULTS; -WORK is used as a temporary work file by the program.

*AMENDS

Contents: The file-amending program.

Purpose: To compare a current version of a line file with the original version and produce a record of the differences. The record contains the MTS command mode lines for transforming the original version into the current version.

Use: The program is invoked by the $RUN command.

Program Key: *EXEC

Logical I/O Units Referenced:

```
0      - the original version of the file.
1      - the current version of the file.
SPUNCH - the MTS command mode lines produced as output.
GUSER  - responses to prompting messages.
SERCOM - prompting messages.
```

If units 0 and/or 1 are not specified, the user will be prompted via SERCOM for the file names; the file names will be read from GUSER.

Description: The *AMENDS program reads two versions of a file: the original version and the current version. Both files must be <u>line</u> files. The entire contents of each line (including trailing blanks) is compared. The comparison is made both on the basis of the line number and the contents of the line. The output to SPUNCH consists of MTS command mode lines such that if

```
$RUN *AMENDS 0=original 1=current SPUNCH=changes
```

is specified, then the following process will convert <u>original</u> into <u>current</u>:

```
$GET original
$SOURCE changes
```

The following sequence illustrates the program.

```
$LIST ORIG
     1     line one
     2     line two
     3     line three
END OF FILE
$LIST NEW
     1     new line one
```

```
          1.25  line one plus
          2     line two
          4     line four
     END OF FILE
     $RUN *AMENDS 0=ORIG 1=NEW SPUNCH=AMENDS
     $LIST AMENDS
          1     $NUMBER 1 .25
          2     new line one
          3     line one plus
          4     $UNNUMBER
          5     3
          6     $NUMBER 4 1
          7     line four
     END OF FILE
```

Hence, if the commands

```
     $GET ORIG
     $SOURCE AMENDS
```

were given, the contents of the file ORIG would be made identical to the contents of NEW.

Two other programs, *DOWNDATE and *UNEDIT, are also available for comparing two versions of a file and producing the changes. A discussion of the differences between these programs is given in the "Files and Devices" section of MTS Volume 1, The Michigan Terminal System (December 1979).

Example: `$RUN *AMENDS 0=OLDFILE 1=NEWFILE SPUNCH=CHANGES`

In the above example, the original file OLDFILE is compared with the current file NEWFILE. The MTS command mode lines are produced in the file CHANGES.

*ANALYZER

Contents:	The XPL grammar-analyzer and syntax-table generation program.
Use:	The program is invoked by the XPL monitor, e.g.,

```
$RUN *XPLGO [I/O units] 0=*ANALYZER
```

Program Key: *EXEC

Logical I/O Units Referenced:

SCARDS	- input grammar description in BNF.
SPRINT	- printed output from analysis.
2	- punched syntax tables output.

Description: The analyzer reads in a BNF description of the grammar for a compiler, and produces as output syntax tables that can be inserted in the skeleton (see *SKELETON in this volume) to build a compiler from this grammar. For instructions, restrictions, and details, see

A Compiler Generator, by McKeeman, Horning, and Wortman, Prentice-Hall, 1970.

*APLEDCL

Contents:	The General Motors APL entity declarations processor.
Purpose:	To process entity declarations for subsequent use by the APL translator and execution-time library.
Use:	The APL entity declarations processor is invoked by the $RUN command.
Program Key:	*EXEC

Logical I/O Units Referenced:

SCARDS - entity declarations source file.
SPRINT - listing.
SERCOM - error comments, if SPRINT is not assigned to *MSINK*.

Parameters: The following parameters must appear in the PAR field of the $RUN command. The parameters must be separated by a comma or by one or more blanks. These parameters are not optional.

APLDCL=FDname	PL/I declarations file. The declarations file may be either a line file or a sequential file.
APLDESC=FDname	master format table file. The master format table file must be a sequential file.

Description: *APLEDCL is a preprocessor for entity declarations to be used by the General Motors APL (Associative Programming Language) system. For a description of that system, see Computing Center Memo 236, "The General Motor’s Associative Programming Language." Also, see the descriptions of *APLGM and *APLLIB in this volume. This entity declarations processor is only part of the MTS implementation of APL, and is not compatible with other programs used for the same purpose at General Motors Research Laboratories.

The significant card columns for the source file on SCARDS are 2 through 72. Columns 1 and 73-80 are ignored.

Example:

```
$COPY ENTITIES
APL_SYSTEM=WHOCARES;
DECLARE 1 PLINE ENTITY(10),
          2 SEGMNT SET,
          2 LNSET MEMBER,
          2 APLATTR,
            3 NAME CHAR(8),
            3 ISEG;

DECLARE 1 SEGA ENTITY(32),
          2 SEGMNT MEMBER,
          2 APLATTR,
            3 TS,
            3 TF,
            3 P(3),
            3 V(3);

DECLARE 1 SEGB ENTITY(56),
          2 SEGMNT MEMBER,
          2 APLATTR,
            3 TS,
            3 TF,
            3 A(3),
            3 B(3),
            3 C(3),
            3 D(3);
END OF FILE
$RUN *APLEDCL SCARDS=ENTITIES SPRINT=*PRINT*
     PAR=APLDCL=DCLFILE APLDESC=MFTFILE
```

In the above example, the entity declarations in the file ENTITIES are processed. The declarations file is DCLFILE and the master format table file is MFTFILE.

*APLGM

Contents:	The General Motors APL Translator.
Purpose:	Compilation of APL source programs.
Use:	The APL translator is invoked by the $RUN command.
Program Key:	*APLGM

Logical I/O Units Referenced:

SCARDS - APL source program(s).
SPRINT - APL and PL/I listings.
SPUNCH - PL/I object modules.
SERCOM - error comments, if SPRINT is not attached to *MSINK*.

Parameters: The following parameters may appear in the PAR field of the $RUN command. The parameters must be separated by a comma or by one or more blanks.

APLDCL=FDname	PL/I declarations file. This parameter is not optional. The declarations file may be either a line file or a sequential file.
APLDESC=FDname	master format table file. This parameter must be specified if the TT option is used (see below). The master format table file must be a sequential file.
APLOBJ=FDname	APL object file (optional). If this is omitted, the APL object program is stored in virtual memory.
ABORT=n	0 - do not perform PL/I compilation. 1 - perform PL/I compilation only if there are no errors <u>or</u> <u>warnings</u> in the APL translation (the default). 2 - perform PL/I compilation only if there are no errors. 3 - perform PL/I compilation only if there are no fatal errors.
PL1PAR=xxx	parameters for *PL1. This parameter, if present, must appear last; the remainder of the parameter list is passed to the PL/I compiler. The *PL1 parameters SM=-(1,71) and DIAG are supplied automatically as required.

Description: This translator is the General Motors APL (Associative Programming Language) translator, modified for use in MTS. For a description of the language, see Computing Center Memo 236, "The General Motor's Associative Pro gramming Language." *APLGM normally invokes the MTS PL/I compiler to complete the translation; for further information, see the *PL1 description in this volume. Use of APL in MTS requires the files *APLEDCL and *APLLIB, which are also described in this volume.

The APL translator translates APL programs into PL/I programs. Both the APL translator and the PL/I compiler are invoked by *APLGM. The APL source program may be (and almost always is) preceded by an options statement, beginning in column 1, of the form:

```
APL option1,option2,...
```

where the available options are:

ARROW	the entity qualification pointer becomes -> instead of >>, which is the default.
TT	the master format table is read in and is made available for optimization purposes, mainly in conjunction with the "?" statement, although certain other statements are optimized automatically if the TT option is present. The default is no format table.
DPRINT	specifies that APL object statements are to be printed between the source statements in the APL listing. The default is no printing of APL object statements.
PACK	specifies that APL object statements are to be packed several per line, as opposed to one per line, which is the default.

Several APL programs may be translated in a single run. They are separated in the source file by %PROCESS statements exactly as is done for *PL1 source programs (see MTS Volume 7, PL/I in MTS). The %PROCESS statement serves merely as a separator for the APL translator, and is passed on without change to the PL/I compiler. The format is

```
%PROCESS('PL/I options');
```

The parameter SM=(1,71) must be included in the PL/I options list. The "%" must be in column one and there cannot be embedded blanks. The %PROCESS statement may be followed by another APL options statement as described above. The standard defaults will be used if no options are given.

The significant card columns for an APL source program are 2 through 72. Column 1 and columns 73-80 are ignored.

Examples:

```
$RUN *APLGM SPUNCH=MYOBJ PAR=APLDCL=DCLFILE,
     APLDESC=MFTFILE APL ARROW,TT

     Source program 1

%PROCESS('SM=(1,71)');
APL DPRINT

     Source program 2

$ENDFILE
```

The above example is a batch run with the source program on cards.

```
$RUN *APLGM SCARDS=MYSOURCE SPRINT=*PRINT*
     PAR=APLDCL=DCLFILE,APLDESC=MFTFILE,ABORT=0,
     APLOBJ=PL1SOU
```

The above example is a terminal run with the source program in MYSOURCE, no PL/I compilation, and the APL object program written into PL1SOU.

```
$RUN *APLGM SCARDS=MYSOURCE SPRINT=*PRINT* SPUNCH=MYOBJ
     PAR=APLDCL=DCLFILE,APLDESC=MFTFILE,PL1PAR=LIST,OPT=2
```

The above example is a terminal run with a PL/I object listing and special optimization requested.

*APLLIB

Contents: The subroutine library for programs compiled by the General Motors APL translator.

Use: *APLLIB is concatenated to the file containing the object modules resulting from an APL compilation on the $RUN command.

Program Key: *EXEC

Description: The APL execution-time library resides in the file *APLLIB. This file has a $CONTINUE WITH *PL1LIB record at the end, and is referenced via the command:

```
$RUN PL/I-obj+*APLLIB [I/O units] PAR=APLDESC=FDname,...
```

The library is described in Computing Center Memo 236, "The General Motor’s Associative Programming Language."

Example:

```
$RUN MYOBJ+*APLLIB SCARDS=TESTDATA PAR=APLDESC=MFTFILE
```

In the above example, the program in the file MYOBJ is executed with the *APLLIB library.

*ASA

Contents: A program to convert lines with ASA printer carriage-control characters in column 1 to lines with machine carriage control in column 1 (which can be directly printed).

Use: The program is invoked by the $RUN command.

Program Key: *ASA

Logical I/O Units Referenced:

SCARDS - the source lines with ASA carriage control.
SPRINT - the output lines with machine carriage control.

Description: The ASA carriage-control functions are defined as follows:

Character	Effect Before Printing
blank	space 1 line
0	space 2 lines
-	space 3 lines
+	suppress line space
1-9	skip to channels 1-9, respectively
A,B,C	skip to channels 10, 11, 12, respectively

The first four ASA carriage-control characters function the same as the MTS logical carriage-control characters except that the line printers do not skip the three bottom lines and the three top lines on a page overflow as done with MTS logical carriage control. The character 1 always skips to the top of the next page (the third line below the perforation). For the other characters 1-9, A, B, and C, see the section "Files and Devices" in MTS Volume 1, _The Michigan Terminal System_.

If the output of *ASA is written to a file, then when the file is ultimately listed on the line printer, the machine carriage-control FDname modifier @MCC must be specified, e.g.,

```
$COPY LISTING *SINK*@MCC
```

Examples:

```
$RUN *ASA SCARDS=-PRINT
```

In the above example, the program lists the file -PRINT on the line printer (*SINK*, the batch default for SPRINT).

```
$RUN *ASA SCARDS=-PRINT SPRINT=LISTING
```

In the above example, the program copies the lines from the file -PRINT to the file LISTING with every first character changed to the appropriate machine carriage-control character.

*ASMG

Contents: The MTS G-Assembler.

Purpose: To assemble System 360/370 Assembly Language programs.

Use: The assembler is invoked by the $RUN command.

Program Key: *ASMG

Alt. Name: *ASMBLR

Logical I/O Units Referenced:

SCARDS	- the source program to be assembled.
SERCOM	- assembler diagnostics.
SPRINT	- listing and reference tables.
SPUNCH	- the resulting object module.
0	- a library of macro definitions. If 0 is not assigned, *SYSMAC will be used.
1	- the resulting object module.
2-5	- additional libraries of macro definitions.
10	- update control cards.

Description: This assembler is the University of Waterloo G-Assembler modified to run under MTS. For a general description of the language, see the IBM publication *IBM System/360 Operating System Assembler Language*, form GC28-6514.

The complete description of the use of this assembler in MTS in given in MTS Volume 14, *360/370 Assemblers in MTS*.

Assembler Options:

The programmer may specify the following options in the PAR field of the $RUN command. The entries may appear in any order and, if any are missing, a standard default will be assumed. Blanks and/or commas are accepted as parameter delimiters and keyword operands may be delimited on the left with equal signs or may be enclosed in parentheses. The minimum acceptable abbreviation of each option is underlined. Alternate forms are also given. For those options that require numeric values, if the value specified is outside the legal range, the appropriate minimum or maximum value will be used.

ALGN or ALIGN / NOALGN or NOALIGN Default: ALGN

If the ALGN option is specified, the assembler flags all alignment errors detected. If the NOALGN option is specified, the assembler only flags alignment errors that involve the fetching of an instruction (e.g., BC or LPSW).

BATCH or MULT / NOBATCH or NOMULT   Default: NOBATCH

If BATCH is specified, the assembler processes a stream of assemblies, the last assembly being terminated with an end-of-file. If NOBATCH is specified, after processing one assembly, the assembler returns to the calling program. See MTS Volume 14, 360/370 Assemblers in MTS, for the complete description of the BATCH option.

CALIGN=n   Default: 0

The CALIGN option may be used to set the column in which the comment field of generated statements is to appear. "n" may be in the range of 0 to 255 with the value of 0 meaning the comment field is to appear starting in the column in which the original comment started. If the desired column is occupied by another field, the comment will start one blank after the end of that field. The first line of a continued statement is considered as columns 1 to 71, the second line as 72 to 127 starting in column 16, etc.

COLUMN=n   Default: 3

The COLUMN option specifies the number of printed columns to be used by the XREF, FULLXREF, UMAP, and RLD listings. The value may range from 0 to 3; the default is 3. For each value of COLUMN, the number of columns used for each of the three listings is as follows:

COL=0	UMAP in 1, XREF in 1, RLD in 3.
COL=1	UMAP in 1, XREF in 1, RLD in 1.
COL=2	UMAP in 2, XREF in 2, RLD in 2.
COL=3	UMAP in 2, XREF in 2, RLD in 3.

DECK / NODECK or ND   Default: DECK

If the DECK option is specified, the object module is produced and written on SPUNCH. If the NODECK option is specified, the object module is not written on SPUNCH.

ESD / NOESD   Default: ESD

If the ESD option is specified, the external symbol dictionary is listed on SPRINT (see also the PRINTER option that controls SPRINT output). If the NOESD option is specified, the external symbol dictionary is not listed on SPRINT.

EXECUTE / NOEXECUTE   Default: NOEXECUTE

If the EXECUTE option is specified, the assembler attempts to load and execute the object module. If the NOEXECUTE option is specified, the assembler does not attempt to load and execute the object module. See MTS Volume 14, 360/370 Assemblers in MTS, for the complete description of the EXECUTE option.

EXTEN / NOEXTEN Default: EXTEN

If the EXTEN option is specified, certain extensions to the OS F-assembler are allowed (see MTS Volume 14). If the NOEXTEN option is specified, strict compatibility with the OS F-assembler is observed. To disable the extended Branch Conditional Register instructions, INSTSET=0 should be specified.

EXTIME=n Default: See text

The EXTIME option specifies the execution time allowed for each job under the EXECUTE option. The value may range from 1 second to 999 seconds. The default is the local or global time limit, if any exists; otherwise, there is no limit. This option is meaningful only if EXECUTE is also specified.

IBLK=n Default: 1

The IBLK option specifies the number of 80-byte records comprising one physical source record on SCARDS. The limits are 1 to 4000. The default is IBLK=1. If IBLK is not 1, the NONUM option is forced.

INSTSET=n or IS=n Default: 70

The INSTSET option specifies the instruction set to be used by the assembler.

IS=0	Instruction set compatible with IBM F-Assembler.
IS=1	Same as F-Assembler with the additional extended branch mnemonics.
IS=9	Instruction set compatible with DOS F-Assembler.
IS=20	Same as IS=60 except all non-Model 20 instructions are deleted and instructions unique to the Model 20 are added.
IS=44	Same as IS=20 except instructions oriented to the Model 44 are added.
IS=60	Same as IS=1 without any System/370 instructions.
IS=67	Same as IS=60 with extra Model 67 instructions added.
IS=70	Same as IS=1 (includes System/370 instructions).
IS=71	Same as IS=70 with VM/CP instructions added.

LINECNT=n or LINE=n Default: 55

The LINECNT option specifies the number of lines to be printed between the headings in the source listing. The limits are 0 to 254; the default is 55. If 0 is specified, the assembler will start a new page only when the EJECT pseudo-op is encountered.

LIST / FULLLIST or FLIST / NOLIST Default: LIST

If the LIST option is specified, a source listing of the assembly's source, macro, and copy sections is printed on SPRINT (see also the PRINTER option that controls SPRINT output). If the FULLLIST option is specified, in addition to the listing produced by the LIST option, each line in every expanded macro is printed as it appears in the macro definition, provided PRINT GEN is in effect in the program. This feature is especially useful when debugging macros. (Note that the FULLLIST option is different from the option of the same name in non-MTS versions of the assembler.) If the NOLIST option is specified, no listing is printed.

LOAD / NOLOAD　　　　Default: NOLOAD

If the LOAD option is specified, an object module is produced and written on logical I/O unit 1. If the NOLOAD option is specified, no object module is written on logical I/O unit 1.

LREF / NOLREF　　　　Default: LREF

If the LREF option is specified, a literal cross-reference listing is printed on SPRINT (see also the PRINTER option that controls SPRINT output). If the NOLREF option is specified, no literal cross-reference listing is printed. The default is LREF.

LSETC=n　　　　Default: See text

The LSETC option may be used to set the default length of a SETC variable (GBLC or LCLC) to a value other than 8. The value may range from 1 to 255 bytes. Space for SETC variables is allocated in a static fashion so that a value of less than 8 will save virtual memory in the assembly but a value of greater than 8 will require extra virtual memory. If NOEXTEN is specified, then 8 will always be used. LSETC will not appear on the header page unless it appears on the parameter list.

MTS / DOS　　　　Default: MTS

If the MTS option is specified, the assembler does not attempt to be compatible with the DOS F-Assembler. If the DOS option is specified, Q- and L-type constants will be flagged as errors and the relocation dictionary will not be sorted. For complete DOS compatibility, NOEXTEN and INSTSET=9 should also be specified.

NUM / NONUM　　　　Default: See text

If the NUM option is specified, the line numbers from which source statements are read are printed in source listings on SPRINT and in the diagnostic information on SERCOM (if TERM is specified). NUM is the default unless either IBLK or OBLK is not 1, in which case NONUM is forced. If the NONUM option is forced, the line numbers are not printed.

OBLK=n   Default: 1

The OBLK option specifies the number of logical records comprising one physical output record on SPRINT. The limits are 1 to 195; the default is OBLK=1. If OBLK is not 1, NONUM is forced, and the logical records are 121 bytes long. If OBLK is 1, the logical records are 133 bytes long.

PRINTER or PRT / NOPRINTER or NOPRT   Default: See text

If the PRINTER option is specified, the Assembler (G) heading page and the trailing diagnostics will be listed on SPRINT. All other listing segments operate under their own parameter control. PRINTER is the default in batch mode and in conversational mode when SPRINT is not assigned to the user's terminal. If the NOPRINTER option is specified, nothing is written to SPRINT, regardless of the settings of the parameters that control listing of different segments of the assembly. NOPRINTER is the default in conversational mode when SPRINT is assigned to the user's terminal.

RENT / NORENT   Default: NORENT

If the RENT option is specified, the assembler checks for violations of program reenterability. If the NORENT option is specified, no checking is performed.

RLD / NORLD   Default: NORLD

If the RLD option is specified, the relocation dictionary is listed on SPRINT. If the NORLD option is specified, the relocation dictionary is not listed.

SIZE=n   Default: 10

The SIZE option specifies the number of pages to be used for the symbol table. The limits are 10 to 64; the default is 10.

STMT / NOSTMT   Default: STMT

If the STMT option is specified, statement numbers are given for flagged statements listed on SERCOM with diagnostic information. This option is used only in connection with the TERM option. If the NOSTMT option is specified, no statement numbers are given for statements listed on SERCOM.

SYSPARM=n

The SYSPARM option specifies the character string value of the system variable symbol &SYSPARM. If the SYSPARM option is not specified, then &SYSPARM will default to a null character string. If EXTEN is not specified, then &SYSPARM may not be referenced and any SYSPARM option will be ignored. SYSPARM will not appear on the header page unless it appears on the parameter list. Commas are not allowed unless the value of the SYSPARM parameter is enclosed in balanced parentheses or single quotes. If it is enclosed in quotes, an embedded quote must be represented by two quotes. The enclosing quotes or parentheses are considered as part of the value, e.g.,

```
PAR=LOAD,SYSPARM=(&AB,('&XY))
```

assigns (&AB,('&XY)) to SYSPARM.

TERM or ERR / NOTERM or NOERR   Default: See text

If the TERM option is specified, flagged lines and assembler diagnostics are listed on SERCOM. TERM is the default in conversational mode. If the NOTERM option is specified, the TERM option is suppressed. NOTERM is the default in batch mode.

TEST or T / NOTEST   Default: NOTEST

If the TEST option is specified, the object module includes the SYM records needed by the Symbolic Debugging System. See the description of the $DEBUG command and the section "Debug Mode" in MTS Volume 1, The Michigan Terminal System, for further information. If the NOTEST option is specified, no SYM records are produced.

UMAP / NOUMAP   Default: UMAP

If the UMAP option is specified, a using map of all registers in USING, DROP, and POP USING statements is printed on SPRINT (see also the PRINTER option that controls SPRINT output). If the NOUMAP option is specified, no using map is printed.

UPCOND=n   Default: 12

The UPCOND option is used only if the UPDATE option is specified. The "n" may range from 1 to 20; the default is 12. Update diagnostic messages ASMG320 and above have been assigned an internal severity code. If the internal code exceeds the UPCOND value "n", the assembly (or assemblies, if BATCH) will be terminated at the start of macro expansion with an ASMG115 diagnostic message.

UPDATE / NOUPDATE   Default: NOUPDATE

If the UPDATE option is specified, an update deck (logical I/O unit 10) and an old master data set (SCARDS) are read simultaneously. The assembly will be done on the updated input. A new master source is not produced. The unit 10 data set may contain ./ DELETE and ./ ENDUP control cards in addition to sequenced update cards as required by the IBM program IEBUPDTE (see the IBM publication, IBM System/360 Operating System: Utilities, form GC28-6586). Other control cards are ignored. The UPDATE option is included primarily for compatibility with non-MTS versions of the G Assembler. MTS users will generally find it more convenient to use *UPDATE for this purpose. However, for large programs, the UPDATE option will be less expensive to use. If the NOUPDATE option is specified, the SCARDS input only is assembled.

UPLIST / FULLUPLIST or FUPLIST / NOUPLIST   Default: UPLIST

If the UPLIST option is specified, changes from logical I/O unit 10 are listed on SPRINT. If the FULLUPLIST option is specified, the updated source from unit 10 and SCARDS is listed on SPRINT. If the NOUPLIST option is specified, no changes from unit 10 are listed on SPRINT. Note that these options are meaningful only if UPDATE has been specified.

URSYM / NOURSYM   Default: URSYM

If the URSYM option is specified, unreferenced symbols are included in the cross-reference listing. If the NOURSYM option is specified, unreferenced symbols are not included in the cross-reference listing. This option is the same as the SHORT or FULL forms of the XREF option.

UTBUFF=n   Default: 3

The UTBUFF option specifies the number of utilities that are to be buffered in virtual memory. Codes 0, 1, 2, and 3 are defined as:

UTBUFF=0	No utilities are buffered.
UTBUFF=1	-SYSUT1 is buffered in virtual memory.
UTBUFF=2	-SYSUT1 and -SYSUT2 are buffered in virtual memory.
UTBUFF=3	-SYSUT1 -SYSUT2, and -SYSUT3 are buffered in virtual memory.

XREF / FULLXREF or FX / NOXREF   Default: XREF

If the XREF option is specified, a condensed cross-referenced symbol table is printed on SPRINT. The COL parameter controls the format of the cross-reference listing (see also the PRINTER option that controls SPRINT output). If the FULLXREF option is specified, a cross-referenced symbol table is produced. If the NOXREF option is specified, no cross-referenced symbol table is produced.

Note: Any of the three XREF options may be used as a keyword with FULL or SHORT as the operand. XREF=SHORT removes from the symbol cross-reference table all entries that are defined but never referenced. (This is the same as specifying just XREF along with NOURSYM.) XREF=FULL produces a cross-refernce table of all symbols used in the assembly. This is the same as specifying just XREF along with URSYM. FULL is the default and is implied if no operand is present for one of the XREF options.

<u>XTOB</u> / <u>NOXTOB</u> Default: NOXTOB

If the XTOB option is specified, all machine instructions that allow both a base register and an index register (RX format) and for which the operand is given as d(b) are assembled with a zero index register and a base register "b". For example, L 1,4(2) assembles as 58102004. If the NOXTOB option is specified, such instructions are assembled with a zero base register and an index register "b". For example, L 1,4(2) assembles as 58120004.

*ASMT

Contents: The TSS Assembler.

Purpose: To assemble System 360/370 Assembly Language programs.

Use: The assembler is invoked by the $RUN command.

Program Key: *ASMT

Logical I/O Units Referenced:

SCARDS	- the source program to be assembled.
SPRINT	- listings and cross reference tables.
SPUNCH	- the assembled object module.
SERCOM	- assembler conversational diagnostics.
GUSER	- conversational source program corrections.
0	- a library of macro definitions.
1	- the assembled object module.
2-5	- additional libraries of macro definitions.

Description: This assembler is the IBM Time Sharing System Assembler modified to run under MTS. It differs from *ASMG primarily in its ability to accept on-line, conversational corrections to the source program as errors are detected. For a general description of the language, see the manuals, IBM System/360 Time Sharing System Assembler Language and IBM System/360 Time Sharing System Assembler Programmer’s Guide, forms GC28-2000 and GC28-2032, respectively.

The complete description of the use of this assembler in MTS in given in MTS Volume 14, 360/370 Assemblers in MTS.

Assembler Options:

The programmer may specify the following options, separated by commas or blanks, in the PAR field of the $RUN command. The options may appear in any order and, if any are omitted, a standard default will be assumed. In the case of conflict, the rightmost entry will be used. The minimum acceptable abbreviation of each option is underlined. Alternate forms are also given.

BATCH or MULT / NOBATCH or NB or NOMULT   Default: NOBATCH

If BATCH is specified, the assembler processes a stream of assemblies, the last assembly being terminated with an end-of-file. If NOBATCH is specified, the assembler returns to system after processing one assembly.

<u>C</u>OPY=FDname

If COPY=FDname is specified, the source program is copied to "FDname". If the WRITE option is specified, all corrections are made to the copy of the source rather than to the original copy.

<u>C</u>OPY

If COPY is specified, the source program is copied to the temporary line file -COPY. If the WRITE option is specified, all corrections are made to -COPY rather than to the original source.

<u>CO</u>RRECT / <u>NOCORRECT</u> or <u>NCO</u> Default: See text

If CORRECT is specified, source statements that produce errors during the first phase of the assembly may be replaced. This allows on-line correction of most source errors. CORRECT is the default in terminal mode. The CORRECT option is not effective if NOERROR is specified. If NOCORRECT is specified, interactive source correction is not operative. This is the default in batch mode.

<u>EC</u>HO / <u>NOECHO</u> or <u>NEC</u> Default: NOECHO

If ECHO is specified, the assembler produces an unedited listing of the source program with line numbers. This listing does not include object code. If NOECHO is specified, the unedited source listing is not produced.

<u>ER</u>ROR / <u>NOERROR</u> or <u>NER</u> Default: ERROR

If ERROR is specified, assembler diagnostics are printed on SERCOM when in conversational mode. If NOERROR is specified, assembler diagnostics are not printed on SERCOM.

<u>ESD</u> or <u>ED</u> / <u>NOESD</u> or <u>NED</u> Default: See text

If ESD is specified, the external symbol dictionary is produced. The default for ESD is the same as that for LIST. ESD is effective only if the LOAD option is specified. If NOESD is specified, no external symbol dictionary is produced. The default for NOESD is the same as that for NOLIST.

<u>L</u>IST / <u>NOLIST</u> or <u>NL</u> Default: See text

If LIST is specified, an edited source listing of the assembly's source, macro, copy sections, and object code is produced. LIST is the default in batch mode and in conversational mode if SPRINT has been specified explicitly or if the PRINT option is specified. The equivalent of the *ASMG FULLIST option may be achieved by issuing the "PRINT FULLGEN" assembler instruction. If NOLIST is specified, no listing is produced. NOLIST is the default in conversational mode when SPRINT has not been specified and the PRINT option is not specified.

L̲OAD=FDname
O̲BJECT=FDname

If the LOAD keyword option is specified, the object module is written on the "FDname". OBJECT is a synonym for LOAD.

L̲O̲A̲D̲ or L̲D̲ / N̲O̲L̲O̲A̲D̲ or N̲L̲D̲
O̲BJECT / N̲O̲O̲OBJECT   Default: LOAD

If neither the above LOAD or OBJECT keyword options is specified, the object module is written on logical I/O unit 1. If logical I/O unit 1 is not specified, the object module is written on SPUNCH. If SPUNCH was also not specified, then the temporary sequential file -LOAD receives the object module. -LOAD is emptied if it already exists. If one desires to add the object module to the end of -LOAD, he may specify LOAD=-LOAD(*L+1). It is not possible to obtain object modules on more than one output unit. OBJECT is a synonym for LOAD. If NOLOAD is specified, no object module is produced. NOOBJECT is a synonym for NOLOAD.

O̲R̲L̲=n   Default: 4096

The maximum object module record length is set to "n". This must be in the range of 32 ≤ n ≤ 4096. The default value is 4096. The minimum of the output device's maximum record length and ORL is used by the assembler.

P̲M̲D̲L̲I̲S̲T̲ or P̲D̲ / N̲O̲P̲M̲D̲L̲I̲S̲T̲ or N̲P̲D̲   Default: NOPMDLIST

If PMDLIST is specified, the program module dictionary is produced. It includes a listing of external symbols, references, and relocation items. If NOPMDLIST is specified, no program module dictionary is produced.

P̲RINT=FDname

If the PRINT keyword is specified, all assembler listings produced will be written on "FDname"; otherwise, the listings will be written on SPRINT.

P̲RINT

If PRINT is specified, the assembler listings produced will be written to *PRINT*. *PRINT* is not held by the assembler, so the user may wish to set AUTOHOLD=ON via the MTS $SET command. A receipt number is produced only if *PRINT* is actually used.

RVAL / NORVAL   Default: NORVAL

If RVAL is specified, an entry point representing the R-value is generated for each regular entry point using the lowercase version of its name. If NORVAL is specified, no R-values are generated.

SCAN

If SCAN is specified, only the syntax scan (phase one) is done. No listings or object modules are produced. Interactive correction of phase one errors is possible.

SOURCE=FDname

If the SOURCE keyword is specified, the source is read from "FDname". If the keyword is not specified, the source is read from SCARDS.

SOURCE

If SOURCE is specified, the assembler source is read from the currently active file or device *AFD*.

SYMTAB or ST / NOSYMTAB or NST   Default: NOSYMTAB

If SYMTAB is specified, a symbol table is printed. The information printed consists of the symbol name, type, length, and value. This is similar to the XREF option except it gives no cross-references. This table is not produced if XREF is specified. If NOSYMTAB is specified, the symbol table is not printed.

SYSMAC=FDname or SM=FDname

If the SYSMAC keyword is specified, the file "FDname" is used as the last macro library to be searched.

SYSMAC or SM / NOSYSMAC or NSM   Default: SYSMAC

If SYSMAC is specified, the TSS-compatible version of *SYSMAC, is used as the last macro library to be searched. If NOSYSMAC is specified, only logical I/O units 0,2,3,4, and 5 are searched as macro libraries. The macro library format is compatible with that produced by *MACUTIL. The order of the macro library search is logical I/O units 2,3,4,5,0 and SM=FDname.

TERM / NOTERM   Default: See text

If TERM is specified, the assembler is forced to behave as if in terminal (conversational) mode. TERM is the default for terminal mode. If NOTERM is specified, behavior is forced as if in batch mode. NOTERM is the default for batch mode.

<u>T</u>EST / <u>NOTEST</u> or <u>NT</u>   Default: TEST

If TEST is specified, the object module will include symbol table (SYM) records to be used by the Symbolic Debugging System (SDS). Currently, entries are produced only for statements that have labels (*ASMG produces entries for DC and DS statements without labels). If NOTEST is specified, no symbol table records are produced.

<u>TESTLIST</u> or <u>TL</u> / <u>NOTESTLIST</u> or <u>NTL</u>   Default: NOTESTLIST

If TESTLIST is specified, an internal symbol dictionary is produced. It lists the information passed to the Symbolic Debugging System through the TEST option. This listing is not produced if the TEST option is not specified. If NOTESTLIST is specified, no internal symbol dictionary is produced.

<u>W</u>RITE / <u>NOWRITE</u> or <u>NW</u>   Default: NOWRITE

If WRITE is specified, statement replacements are made to the actual source if it is a line file. WRITE is effective only if CORRECT is specified. If the COPY option is specified, corrections are made to the copy rather than to the source. If NOWRITE is specified, source files are not changed when replacement lines are given to the assembler.

<u>X</u>REF / <u>NOXREF</u> or <u>NX</u>   Default: See text

If XREF is specified, a cross-referenced symbol table is produced. The default for XREF is the same as that for LIST. If NOXREF is specified, no cross-referenced symbol table is produced. The default for NOXREF is the same as that for NOLIST.

*BCDEBCD

Contents: The BCD-to-EBCDIC conversion program.

Purpose: To convert cards punched on an IBM 026 keypunch (BCD) to the equivalent card codes as if punched on an IBM 029 keypunch (EBCDIC).

Use: The program is invoked by the $RUN command.

Program Key: *BCDEBCD

Logical I/O Units Referenced:
SCARDS - BCD lines to be converted.
SPRINT - the listing of converted lines.
SPUNCH - EBCDIC lines resulting from the conversion.

Description: The following conversion is applied to all input lines:

BCD Card Code	Character	EBCDIC Card Code
0-4-8	(%	12-5-8
12-4-8	) ¤	11-5-8
4-8	′ @	5-8
12	+ &	12-6-8
3-8	= #	6-8

All other card codes are left unchanged. For the two characters given in the character column, the first represents the scientific symbol and the second represents the commercial symbol.

This conversion maps those characters that have a dual symbolism (scientific and commercial) on the 026 keypunch to the appropriate 029 keypunch scientific code. Note that there is no way to represent the following characters:

% ¤ @ & #

as this program will convert them to their scientific equivalents.

Example:

```
$RUN *BCDEBCD SPUNCH=EBCDFILE
```

In the above example, the BCD source lines are read from *SOURCE* (the default for SCARDS) and the converted EBCDIC lines are written to the file EBCDFILE.

*COBOLU

Contents: The IBM American National Standard (ANSI) COBOL compiler.

Purpose: To compile ANSI COBOL programs (for the language accepted, see the IBM OS Full American National Standard COBOL System Library Manual, form GC28-6396). This manual describes four COBOL compilers; the compiler used in MTS is Version 2, 360S-CB-545.

Use: The compiler is invoked by the $RUN command.

Program Key: *COBOLU

Logical I/O Units Referenced:

SCARDS	- the COBOL source program to be compiled.
SPRINT	- source program listing, diagnostics, etc.
SPUNCH	- the resulting object module (controlled by DECK parameter).
0	- the resulting object module (controlled by LOAD parameter). Defaults to -LOAD.
SERCOM	- error and informational messages.

Parameters: The following parameters may be specified in the PAR of the $RUN command. The parameters may be specified in any order and must be separated by commas. A default will be assumed for omitted parameters. The underlined portion of each parameter may be used as an abbreviation. There may be no embedded blanks in the parameter string.

APOST or QUOTE

The APOST parameter specifies that the compiler is to accept ’ as the character used to delimit literals. QUOTE specifies that " is to be used. The default is APOST.

CLIST or NOCLIST

The CLIST parameter specifies that a condensed object listing is to be written on SPRINT. The procedure portion of this listing will contain register assignments, source line numbers, verbs, and the location of the first generated instruction for each verb. The CLIST and PMAP parameters are mutually exclusive. The default is CLIST.

<u>DEC</u>K or <u>NODEC</u>K

The DECK parameter specifies that the resulting object module is to be produced on SPUNCH. The default is NODECK.

<u>DMA</u>P or <u>NODMA</u>P

The DMAP parameter specifies that a map of all data names used in the program is to be written on SPRINT. This map is necessary for locating variables in a dump. The default is DMAP.

F<u>LAGW</u> or F<u>LAG</u>E

The FLAGW parameter specifies that all diagnostics are to be written on SPRINT. The FLAGE parameter specifies that warning level (level W) messages are not to be written on SPRINT. The default is FLAGW.

<u>LIB</u> or <u>NOLIB</u>

The LIB parameter specifies that BASIS and/or COPY statements may be in the source program. If COPY and/or BASIS statements are not present, specifying the NOLIB parameter will provide more efficient compiler processing. The default is NOLIB.

LINE<u>CNT</u>=n

The LINECNT parameter specifies the number of lines to be written on each page of the source program listing. This must be in the range $1 \le n \le 99$. The default is 57.

<u>LOA</u>D or <u>NOLOA</u>D

The LOAD parameter specifies that the resulting object module is to be produced on logical I/O unit 0, which the compiler will default to the file -LOAD if not assigned. The default is LOAD.

<u>PMA</u>P or <u>NOPMA</u>P

The PMAP parameter specifies that a full listing of the object code is to be written on SPRINT. The PMAP and CLIST parameters are mutually exclusive. The default is NOPMAP.

<u>SEQ</u> or <u>NOSEQ</u>

The SEQ parameter specifies that the compiler is to check the sequence field (columns 1 - 6) of the source statements. A warning message will be produced for all source statements out of sequence. The default is NOSEQ.

SIZE=nnnnnn

The SIZE parameter specifies the amount (in bytes) of virtual memory to be made available to the compiler. The default is 141312.

SOURCE or NOSOURCE

The SOURCE parameter specifies that a source program listing is to be written on SPRINT. The default is SOURCE.

SPACE1, SPACE2, or SPACE3

This parameter specifies the type of spacing to be used in the source program listing when SOURCE is specified. SPACE1 specifies single-spacing, SPACE2 specifies double-spacing, and SPACE3 specifies triple-spacing. The default is SPACE1.

SUPMAP or NOSUPMAP

The SUP parameter specifies that suppression of the object module generation and the object code listing is to take place if a level E compilation error occurs. The default is SUPMAP.

TRUNC or NOTRUNC

The TRUNC parameter specifies that the arithmetic item of a COMPUTATIONAL field is to be truncated to the number of digits specified in the PICTURE clause when moved. If NOTRUNC is specified, the movement of the item is dependent on the size of the field (halfword, fullword). The default is NOTRUNC.

VERB or NOVERB

The VERB parameter specifies that procedure-names and verb-names are to be listed with the associated code in the object listing. VERB is effective only if PMAP or CLIST are specified. The default is VERB.

XREF or NOXREF

The XREF parameter specifies that a cross-reference listing is to be written on SPRINT. The default is NOXREF.

ZWB or NOZWB

The ZWB parameter specifies that the compiler is to generate code to strip the sign from a signed external decimal field when comparing this field to an alphanumeric field. The signed external decimal field is moved to an intermediate field, in which its sign is removed, before it is compared to the alphanumeric field. The default is ZWB.

Description: The compiler in *COBOLU is the OS/360, Release 21.6, Version 2 of the IBM ANSI COBOL compiler, with an interface program to allow it to execute under MTS.

Because this compiler produces object modules normally designed to run under OS/360, these object modules usually cannot be run directly under MTS. An environment designed to simulate the OS environment is provided by the program *FAKEOS (see the description of *FAKEOS in this volume). The object modules produced by the compiler contain OS/360 SVCs and control blocks and will not execute properly in MTS.

When running a program produced by the compiler, it is also necessary to specify *COBLIB as the subroutine library from which the loader is to resolve external references. This file should be concatenated to the file(s) containing the program to be run when *FAKEOS is invoked.

The compiler in *COBOLU is taken directly from the OS/360 ANSI COBOL compiler. Therefore, any programs which will compile successfully under OS/360 ANSI COBOL will compile successfully (i.e., without diagnostics) under *COBOLU. However, the current version of the execution-time support environment in *FAKEOS does not provide support for all the services which object modules generated by *COBOLU may request.

In particular, the current version of *FAKEOS only supports the access methods BSAM, QSAM, and part of BDAM. Therefore, programs which request the DIRECT, RELATIVE, and INDEXED types of file organizations will compile correctly but will not necessarily execute successfully. Requests for object program SEGMENTATION are also currently not supported.

The rules for determining file characteristics (as contained in the DCB - the Data Control Block) are essentially the same as in OS/360. Namely, if information on file characteristics is given at compile-time (through the SELECT and FD statements), it will be assembled into the DCB and need not be specified at execution-time.

However, if information on file characteristics is omitted at compile-time, it must be specified at execution-time. In particular, this applies to COBOL programs in the following way:

(1) In the SELECT statement, the device code should be omitted, e.g.,

```
SELECT FILE1 ASSIGN TO UT-S-DDNAME1
```

However, the device information should not be specified at execution-time, and is one exception to the rule that information not specified at compile-time must be specified at execution-time.

(2) Information on record format and logical record length is always determined at compile-time by looking at the FD (file description) for each file. If there is only one record entry for a file, then the record format (RECORDING MODE) is always F (FIXED) and the logical record length is the length of the record. If there is more than one record described for a file and all have the same length, then the above rule is in effect. If the records have different lengths, then the record format (RECORDING MODE) is V (VARIABLE) and the logical record length is the length of the longest record (plus 4 for the record length control field). If the BLOCK CONTAINS clause is omitted, then the blocksize defaults to the logical record length.

(3) If BLOCK CONTAINS 0 RECORDS is specified, then the blocksize entry in the DCB is left blank at compile-time. Under this condition, the record format modifiers must be specified for that file in part II of the parameter string for the execution-time environment for *FAKEOS (see the *FAKEOS description for further details).

(4) *FAKEOS handles variable records in several different ways, depending on the device type to which they are assigned. If the Data Control Block for a COBOL "file" specifies that the records are variable length, and the ddname for that file is assigned to a unit-record device (or pseudodevice equivalent), *FAKEOS will use the information in the variable length record control field for unblocking and formatting purposes, but the control field will not be sent to the device. If a variable length record is sent to a non-unit-record device (i.e., an MTS file or tape),

records will be written to that device with the control field and blocked as specified.

(5) *FAKEOS ignores the label record field of the DCB. However, the compiler insists that a LABEL RECORDS clause be present in every file description (FD). Therefore, LABEL RECORDS ARE STANDARD should be specified for every file.

The file *COBLIB contains the subroutine library for COBOL-compiled programs. Since many of these subroutines rely on OS/360 services, they cannot be used by programs that run directly under MTS. These programs must be run under the control of *FAKEOS. *COBLIB should be concatenated on the $RUN command to the object module resulting from the COBOL compilation, e.g.,

```
     $RUN *FAKEOS PAR=E=object+**COBLIB
```

The following example MTS COBOL job illustrates many of the above points.

```
$RUN *COBOLU
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  EXAMPLE.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER.  IBM-360-67.
       OBJECT-COMPUTER.  IBM-360-67.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT FILE1 ASSIGN TO UT-S-DDNAME1.
           SELECT FILE2 ASSIGN TO UT-S-DDNAME2.
           SELECT FILE3 ASSIGN TO UT-S-DDNAME3.
       DATA DIVISION.
       FILE SECTION.
       FD  FILE1
           RECORD CONTAINS 80 CHARACTERS,
           RECORDING MODE IS F,
           BLOCK CONTAINS 0 RECORDS,
           LABEL RECORDS ARE STANDARD.
       01  IN-RECORD1 PIC X(80).

       FD FILE2
           RECORD CONTAINS 60 TO 80 CHARACTERS,
           RECORDING MODE IS V,
           BLOCK CONTAINS 1 RECORDS,
           LABEL RECORDS ARE STANDARD.
       01  IN-RECORD2A PIC X(60).
       01  IN-RECORD2B PIC X(80).

       FD  FILE3
           RECORD CONTAINS 10 TO 30 CHARACTERS,
           RECORDING MODE IS V,
           BLOCK CONTAINS 0 RECORDS,
           LABEL RECORDS ARE STANDARD.
       01  IN-RECORD3A PIC X(10).
       01  IN-REOCRD3B PIC X(30).
       PROCEDURE DIVISION.

         .
         .
         .

$ENDFILE
$RUN *FAKEOS PAR=E=-LOAD+*COBLIB;DDNAME1=FDNAME1@F(80,80)
    DDNAME2=FDNAME2 DDNAME3=FDNAME3@VB(344,34)
```

In the case of FILE1 (DDNAME1), RECORD CONTAIN 0 RECORDS was specified, therefore, the record format modifier @F(80,80) was specified at execution-time. For FILE2 (DDNAME2), all information was specified at compile-time, therefore, no record format modifiers were needed at execution-time. For FILE3 (DDNAME3), as in FILE1, RECORD CONTAINS 0 RECORDS was specified, therefore, the record format modifier @VB(344,34) was specified. Note that this information is a minimum blocking factor of 10. The maximum logical record length is 30 characters (plus 4 for the logical record control field). The blocksize is 10 times this plus 4 for the block control field.

See the description of *FAKEOS for detailed information on record format modifiers.

*COINFLIP

Contents:	The coin-flip demonstration program.
Use:	The program is invoked by the $RUN command, e.g., $RUN *COINFLIP
Program Key:	*COINFLIP
Description:	The program is self-describing. It is intended for terminal use.

*DCALC

Contents:	A sophisticated desk-calculator program.
Use:	The program is invoked by the $RUN command.
Program Key:	*DCALC
Description:	This program is intended to be used for desk calculator type work in conversational mode. For an elementary description of the program, run it and enter the command "?". For additional information, enter the command "#". The command "/T" will stop the program.

*DOWNDATE

Contents:	A program to down-date a file.
Purpose:	To compare a current version of a program with the original version and produce *UPDATE control cards for transforming the original version into the current version.
Use:	The program is invoked by the $RUN command.
Program Key:	*DOWNDATE

Logical I/O Units Referenced:

GUSER	- prompting responses.
SERCOM	- prompting messages for the FDnames and error comments.
SPUNCH	- *UPDATE control cards produced as output.
0	- original version of program.
1	- current version of program.
SPRINT	- PAR=LIST output

Parameters: The following parameters may be specified in the PAR field of the $RUN command. The parameters must be separated by commas.

MAXDEL=n	"n" is an estimate of the maximum number of lines to be deleted in any one deletion from the original version of the file. Specifying a value slightly larger than the number of lines actually deleted will greatly reduce the cost of running the program. If "n" is too small, the output written to SPUNCH will be valid, but much longer than necessary. The default is plus-infinity.
MAXCOMP=n	"n" is the maximum number of lines to be compared between the original and current versions before deciding that a "resynch" has been found. The actual number of lines compared depends on the size of the preceding deletion and the value of the BREAK and SLOPE parameters. The default is 25.

BREAK=n
SLOPE=m
	"n" and "m" control the number of lines that must match between the original and current files before a "resynch" is recognized. If "d" is the size in lines of the deletion preceding the resynch, then the match must be "n+(d-n)/m" lines long, unless "d<n", in which

	case, "d" is used. If the result is greater than the value of MAXCOMP, then MAXCOMP is used. The defaults are n=3 and m=7.
LIST	A listing of the differences between the old and new versions will be printed on SPRINT.

Description: The *DOWNDATE program reads two versions of a file: the original version and the current version. Both files are read sequentially, and may be either magnetic tape files, sequential files, or line files. All line numbers are ignored. The output from *DOWNDATE consists of a set of *UPDATE control cards which will transform the original version of the file into the current version.

If units 0 and/or 1 are not assigned, the program will prompt for the FDnames of the two files via the logical I/O unit SERCOM; the FDnames of the two files will be read from GUSER.

The two versions of the file are assumed to contain 80-character logical records. Only the first 72 columns are compared; columns 73-80 of the original version are assumed to contain *UPDATE compatible sequence-IDs; columns 73-80 of the current version are ignored.

For a description of the *UPDATE control cards, see the *UPDATE description in this volume. Two other programs, *UNEDIT and *APC, are also available for comparing two versions of a file and producing the changes. A discussion of the differences of these three programs is given in the "Files and Devices" section of MTS Volume 1, _Michigan Terminal System_.

Example:

```
#$run *downdate spunch=changes
#EXECUTION BEGINS
 WHERE IS OLD SOURCE?
 oldfile
 WHERE IS CURRENT SOURCE?
 newfile
#EXECUTION TERMINATED
```

In the above terminal example, the original file OLDFILE is compared with the current file NEWFILE. The *UPDATE control cards are produced in the file CHANGES. The input from the user is in lowercase and the output from the program is in uppercase.

*EBCDBCD

Contents:	The EBCDIC-to-BCD translation program.
Purpose:	To translate lines from EBCDIC to 7090 Augmented BCD.
Use:	The program is invoked by the $RUN command.
Program Key:	*EBCDBCD

Logical I/O Units Referenced:

SCARDS - EBCDIC lines to be translated.
SPRINT - listing of the lines resulting from translation.
SPUNCH - augmented BCD lines resulting from translation.

Description: Records are read from SCARDS, translated according to the table shown on the next page, and written to SPUNCH. Each output line is also prefixed by a blank character (for carriage control) and written to SPRINT. An end-of-file on SCARDS terminates the program.

Example:

```
$RUN *EBCDBCD SCARDS=OLDFILE SPUNCH=NEWFILE
```

In the above example, EBCDIC lines are read from the file OLDFILE, translated to BCD, and written to the file NEWFILE.

The following translation is applied to each character of each input line:

EBCDIC Input Character	Card Code	BCD Output
Digits		Unchanged
Uppercase letters		Unchanged
Lowercase letters		Uppercase equivalents
. , $ * - / ; | ¬ _ "		Unchanged
+	12-6-8	12
&	12	0-7-8
(	12-5-8	0-4-8
)	11-5-8	12-4-8
'	5-8	4-8
=	6-8	3-8
<	12-4-8	12-6-8
>	0-6-8	6-8
%	0-4-8	2-8
?	0-7-8	12-0
:	2-8	5-8
#	3-8	0-2-8
@	4-8	11-0
All others		Blank

*ENCODE, *DECODE

Purpose: To provide compression/blocking procedures for file with line-printer records.

Use: This program is invoked by the $RUN command.

Program Keys: *ENCODE
*DECODE

Logical I/O Units Referenced:
SCARDS - input records.
SPRINT - output records.
SERCOM - summary statistics (*ENCODE only).

Description: The ENCODE program reads line-printer records (with either logical and/or machine carriage-control characters) from SCARDS and compresses them into output records with a maximum size of 255 bytes per line. Normally, several line-printer records will fit into one output record transmitted to SPRINT. This type of compression can be used to compact print-files prior to transmission to remote batch-station equipment.

When an end-of-file is encountered on SCARDS, summary statistics are printed on SERCOM. These statistics give the printing time calculated for a 315 line-per-minute printer operating with the Data Concentrator DUPLEX transmission protocol over a 2000 bit-per-second data set.

Note: Automatic carriage-control conversion to machine carriage control can be disabled by specifying PAR=NOCC on the $RUN command for ENCODE.

The DECODE program decompresses and deblocks compressed input records read from SCARDS and transmits them to SPRINT. Input records may be of any length up to 32767 bytes.

Examples:

```
$RUN *ENCODE SCARDS=INPUTFILE SPRINT=OUTFILE
```

In the above example, the line-printer records are read from the file INPUTFILE, compressed, and written to the file OUTFILE.

```
$RUN *ENCODE SCARDS=INPUTFILE SPRINT=OUTFILE PAR=NOCC
```

This example is the same as above except that automatic machine carriage-control conversion is disabled.

```
$RUN *DECODE SCARDS=INPUTFILE SPRINT=OUTFILE
```

In the above example, compressed input records are read from the file INPUTFILE, decompressed, and written to the file OUTFILE.

*FAKEOS

Contents: The OS/360 environment simulation program.

Purpose: To allow object modules containing OS/360 SVCs, control blocks, and references to OS/360 I/O access methods to execute under MTS.

Use: The OS/360 environment is invoked by the $RUN command.

Program Key: *FAKEOS
*FAKEOSCREBL for *FAKEOSCREBLDL
*FAKEOSTRACE for *FAKEOSTRACE

Logical I/O Units Referenced:

SERCOM	- error messages from *FAKEOS and messages from the Write-to-Operator (WTO) SVC.
GUSER	- replies to the Write-to-Operator with Reply (WTOR) SVC and to prompting by *FAKEOS.
17	- default for WTL (Write-to-Log) SVC output.

*FAKEOS does not reference directly any other logical I/O units; however, programs running under *FAKEOS may. The logical I/O units that can be used by programs running under *FAKEOS are:

SCARDS	- default assignment for SYSIN.
SPRINT	- default assignment for SYSPRINT and SYSOUT.
SPUNCH	- default assignment for SYSPUNCH.
0	- default assignment for SYSLIN.

Description: *FAKEOS acts as a monitor. It intercepts all user SVCs, OS-type I/O calls, and program interrupts and attempts to service them. Programs run under *FAKEOS may make direct use of MTS services but all SVCs intercepted will be interpreted as OS/360 SVCs.

Parameters:

The parameter list passed to *FAKEOS consists of three parts separated by semicolons. The first part is mandatory; the other two parts are optional. The first part consists of *FAKEOS options, the second part contains the input/output assignments, and the third part specifies what parameter string (if any) is passed to the initial user program module. If no parameters are passed to FAKEOS, it will read them from GUSER. All but the last parameter line read from GUSER must end with the current MTS continuation character (initially "-").

Part I - *FAKEOS Options

There are a great many options. In the following, these options are grouped in categories so that all the core-management options will be together, all the dynamic loading options will be together, etc. An asterisk preceding an option means that the casual user of *FAKEOS will probably need to use the option; no asterisk means that the option is not of very general interest but is included for the sake of completeness.

Dynamic Loading Options

In this section, xxx refers to the eight-character member name referenced by a call to LOAD, LINK, or XCTL.

*(1) The most important option is the "what-to-load-first" option. This option specifies the FDname containing the initial user object module. It takes the form of either:

```
E=FDname
E='FDname'
FDname
```

"FDname" may be a concatenation of files or devices. It should be complete in the sense that one should be able to use the MTS command

```
$LOAD FDname
```

without obtaining loading errors. If the third form is used, then the what-to-load-first parameter must appear as the very first option in the *FAKEOS options list. If the what-to-load-first option does not appear, then the member option MEMBER=member_name *must* appear as well as the specification of either JOBLIB or STEPLIB in the second parameter list. For example, if one had an object module in the file -LOAD and it contained a COBOL program that needed subroutines in the system library *COBLIB, one would specify

```
E=-LOAD+*COBLIB
```

(2) The "load-segment" option takes the form

```
LO=n
```

where "n" is 0 or any valid user segment number, currently 6-12 (defaults to 7). This option specifies into which segment user program modules are to be loaded. If 0 is specified, any available segment will be used.

(3) The "load-prefix" option takes the form

    P=ccc

where "ccc" is a character string of length one to ten. Upon interception of a LOAD, LINK, or XCTL request, *FAKEOS will normally attempt to load the requested object module from the file cccxxx where ccc is as given in the load-prefix keyword and xxx is the eight-character member name to which the user request refers. If the prefix is not specified, then the user's own "ccid" is implied (i.e., the files to be loaded are assumed to be owned by the running "ccid"). The load-prefix option may also be specified as a parenthesized list; i.e., "P=(ccc1,ccc2,...,cccn)". The simple unparenthesized form is equivalent to the parenthesized form with only one entry; i.e., ccc1 only. When *FAKEOS uses the load-prefix option, it will check to see if there is a file name of the form ccc1xxx, where xxx is the eight-character member name. If that file does not exist, a check will be made to see if a file name of the form ccc2xxx exists, etc. The first existent file will be the one from which the object module will be loaded. A null sublist entry in the load-prefix option (denoted by "(," at the beginning, ",," in the middle, or ",)" at the end) denotes that the current running "ccid" should be used as a prefix at that point (i.e., "ccid:" will be the five-character prefix used). This option interacts with the library (L=) option, the library-resolve-first (LRF) option, and the specification of the DDNAMEs JOBLIBn and STEPLIBn. This option has no effect on the what-to-load-first option. See a more detailed description below of how LINK, LOAD, and XCTL requests are resolved.

(4) The library option is used to specify an object module library from which object modules are to be extracted to resolve user LOAD, LINK, and XCTL requests. It is specified in the form

    L=FDname

where "FDname" specifies the file or device to be used as a library. If the load-prefix option P is also given, this may result in actually loading from the explicitly concatenated FDname cccxxx+yyy, where ccc is as given in the load-prefix option, xxx is the OS member name requested in the call to LOAD, LINK, or XCTL, and yyy is the FDname specified in the library option (i.e., L=yyy). Note that it is not necessary that yyy be a library; it may be simply an object module that must be loaded everytime a LOAD, LINK, or XCTL request is made. This option interacts with the library-resolve-first LRF option and with the specification of the DDNAMEs JOBLIBn and STEPLIBn. This option has no effect on the what-to-load-first option. This option requires that the loader symbol tables be available. The global MTS SYMTAB option will be set ON if it was OFF, and the user will be notified that this happened; at the end of the run, SYMTAB will be reset to OFF status.

(5) The member option, which is given in the form

```
MEMBER=member_name
```

is a alternate to the what-to-load-first option given above. "member_name" is (at most) eight characters in length and will be resolved from a library attached to the DDNAMEs JOBLIB and STEPLIB in the second parameter list.

(6) The library-resolve-first option, which is given in the form

```
LRF
```

informs *FAKEOS that it is to attempt to resolve the member <u>xxx</u> from a library before it attempts to resolve <u>xxx</u> from the file <u>cccxxx</u> (where <u>ccc</u> is any of the optionally given load-prefixes). *FAKEOS will attempt to resolve from an L= library before it attempts to resolve from a library attached to the DDNAMEs JOBLIBn and STEPLIBn.

(7) The suppress-loading option, which is given in the form

```
NL
```

specifies that member names referenced in requests to LOAD, LINK, and XCTL are to be resolved from the object module initially loaded (where they are found as entry points or csect names). This option allows one to load an entire subsystem or processor (e.g., RPG) at one time rather than actually loading object modules at each LOAD, LINK, or XCTL request. The specification of the suppress-loading option allows only one reference to each member name since modules are not unloaded and the processor in question might require that a refreshed image of the module be loaded. This option requires that the loader symbol tables be available. The global MTS SYMTAB option will be set ON if it was OFF, and the user will be notified that this happened; at the end of the run, SYMTAB will be reset to OFF status.

Resolving Requests for LINK, LOAD, and XCTL

When a request is made to load an object module via LINK, LOAD or XCTL, *FAKEOS will either try to find that object module in a library or in a file (with somewhat the same name as the object module). If the library-resolve-first (LRF) option was specified, then *FAKEOS will attempt to find the module in a library first; if unsuccessful in doing this, it will then attempt to find the module in a file. If the library-resolve-first option is not chosen (the default), *FAKEOS will attempt to find the module in a file before it attempts to find it in a library. In either case, the following is true. When attempting to find the module in a library, it will first attempt to find it in the "L=" library, then in a step library, and lastly in a job library. When looking for a module in a step or job library, FAKEOS will first try the library assigned to the DDNAME to STEPLIB or JOBLIB. If the module is not found in this library, it will look in all libraries attached to a DDNAME of the form STEPLIBn or JOBLIBnn where "n" is any character. For example, the order of search might be "L=" library, STEPLIB, STEPLIBA, JOBLIB, or, JOBLIBA. In searching for a file (as opposed to a library), FAKEOS will use the load-prefixes in the order specified and will prefix the module name with each prefix. When FAKEOS finds an existing file for one of the prefixed filenames, it will attempt to load from that file.

Core Management Options

*(1) The core-limit-size option allows one to specify the upper limit to the amount of virtual memory (in bytes) that the user may explicitly acquire via a GETMAIN request. This option may take either of the following forms:

```
C=nK
C=n
```

where "n" is a decimal number string (e.g., C=30K). The K indicates a multiplicative factor of 1024; thus C=2K is equivalent to C=2048. Unless the OS-core-limit-size option (see next parameter description) is specified, a call is made to the system subroutine GETSPACE for each GETMAIN request and a call is made to the system subroutine FREESPAC for each FREEMAIN request. It is recommended that the user utilize this option when running programs that contain variable GETMAINs since a user pays for all the virtual memory he requests (and gets).

(2) The OS-core-limit-size option is similar to the core-limit option described above. This option has two forms:

```
OSC=nK
OSC=n
```

with the same interpretation as for the core-limit-size option given above. With this option, *FAKEOS actually acquires a virtual memory space of the requested size and allocates explicit GETMAIN requests from that space. This allocation is done in a manner similar to that used by OS/360 in that space is allocated from the highest possible free area in the region. Some processors (notably ICES) have built into them the knowledge of how OS/360 allocates memory and they use this fact in making certain decisions. With this option, only one call is ever made to GETSPACE to actually acquire virtual memory to be used for explicit user GETMAIN requests. *FAKEOS has its own memory allocation scheme to acquire (GETMAIN) and release (FREEMAIN) blocks within the entire region.

(3) The get-memory-segment option takes the form

```
G=n
```

where "n" is 0 or any valid user segment number, currently 6-12 (defaults to 7). This option specifies in which segment *FAKEOS is to acquire virtual memory in response to an explicit GETMAIN request. If the OS-core-limit-size option is given, then the initial block of virtual memory is acquired in the segment given in the get-memory-segment option. If 0 is specified, any available segment will be used.

(4) The system-memory-segment option takes the form

```
S=n
```

where "n" is 0 or any valid user segment number, currently 6-12 (defaults to 7). This option specifies in which segment *FAKEOS is to acquire virtual memory in response to an implicit request (such as for IOBs, I/O buffers, device tables, DEBs, STAE control blocks, and PIEs (program interrupt elements)). If 0 is specified, any available segment will be used.

Program Interrupt Processing Options

*FAKEOS tries to take corrective action when a program interrupt occurs in user mode. The interrupt must fit all of the following conditions:

(a) type 4 (protection) or type 5 (addressing),
(b) the interrupt must be precise,
(c) the subject instruction must not be attempting to alter memory outside the user's allocated space which is defined to be the get-memory-segment (segment 7 by default) and the load-segment (again, segment 7 by default).
(d) the instruction actually causing the interrupt must be in the user's load-segment (defaults to segment 7 unless specified otherwise by the load-segment option) or in the user's get-memory-segment (defaults to segment 7 unless specified otherwise by the get-memory-segment option),
(e) the offending address (the address forcing the interrupt) must be in segment 0.

The user may specify that *FAKEOS should use corrective action on all such interrupts, on only a given subset of those interrupts, or only a certain number of such interrupts. *FAKEOS may decide that corrective action will not be taken (see the paragraph at the end of this section).

(1) The no-program-interrupt option, which is given as

```
NOPGNT, NOPGNTS, or NC
```

disables this corrective action entirely and also inhibits all SPIE exits.

(2) The CVT-only option, which is given as

```
CVT or NP
```

specifies that only the CVT (location X'0010') is allowed to be referenced. Some OS programs need to get to the CVT. *FAKEOS maintains a CVT for this purpose and when location X'0010' is referenced, *FAKEOS will take the appropriate action by replacing the reference to location X'0010' by a reference to a location containing the *FAKEOS CVT address and reexecuting the instruction.

(3) The controlled-count option is specified by either of the keywords

```
PGNT=nnn
PGNTS=nnn
```

where "nnn" is the maximum number of program interrupts of the following type upon which *FAKEOS is to take corrective action: the bad operand does not reference the word at location X'0010' (the CVT), or the word at X'0050' (the timer), or a valid user location. Note this option implies unlimited references to the CVT or timer locations.

(4) The unlimited-count option is specified by

```
PGNT or PGNTS
```

This is really the same as the controlled-count option but a default value of 9999999 is used. Note that use of this option implies unlimited reference to the CVT or timer locations.

(5) The OS-low-core-image option, which is given as

```
OSLC
```

informs *FAKEOS that the environment of the program is to be established in such a way that locations X'0000' to X'07FF' appear to be the same as they would be in a real OS/360 system. If a program interrupt occurs due to a reference to these locations, the instruction will be reexecuted referencing a copy of these locations taken from a real OS/360 system. If this option is _not_ used, the corrective action will be to reexecute the offending instruction with offending addresses changed to reference the *FAKEOS CVT location or random locations in *FAKEOS itself.

(6) The Unit Control Block option, which is given as

    UCBS

informs *FAKEOS that the program intends to use the UCB look-up table in the CVT to find a UCB. *FAKEOS will attempt to make this work.

If *FAKEOS decides not to take corrective action for any reason whatsoever and the offending instruction is located in the user load-space or get-memory-space and the no-program-interrupts option was not specified, then an exit is taken to the user's SPIE routine (if it exists). If the SPIE exit has not been set up, then ABEND processing begins. Thus if a no-program-interrupts option is specified, the user will never get a SPIE exit but may get a STAE exit.

BLDL Options (for BPAM libraries)

When a user program issues a BLDL macro, *FAKEOS normally reads a file that contains information to be returned to the user as a result of the BLDL call. *FAKEOS uses the file that is attached to the DDNAME LINKLIB specified in the second parameter list if no DCB parameter was specified on the BLDL macro or uses the file that the DCB refers to if a DCB parameter was specified. The file so read is constructed (at present) by a *FAKEOS utility program called *FAKEOSCREBLDL.

(1) The BLDL-count option is specified in the form

    BLDL=n

where "n" is a decimal number. This option specifies that before the user program is to start, *FAKEOS is to read the file attached to the DDNAME LINKLIB into memory. *FAKEOS will construct an in-core BLDL list of (at most) "n" entries using the data so read. The file will then be released and LINKLIB will become undefined. When the user program issues a BLDL, *FAKEOS will use the in-core BLDL list that it constructed.

(2) The BLDL-entry-point option is specified in the form

    BLDL=symbol

where "symbol" is an entry in a special low-core symbol table. The symbol FAKEBLDS in the system low-core symbol table LCSYMBOL will point to this special low-core symbol table. If "symbol" actually appears in this special low-core symbol table, the corresponding address (if nonzero) will point to a BLDL list constructed to the same internal specifications as *FAKEOS would have used if it had constructed the BLDL table as given in (1) above. This option is useful for systems programmers only.

Options Affecting the Defaulting of DDNAMEs

When the user program requests a specified DDNAME to be opened but that DDNAME has not been assigned in the second parameter list, *FAKEOS tries to assign a standard default for that DDNAME. If it cannot apply a standard default, the user may be prompted to enter a DDNAME-FDname assignment. The user will never be prompted for default assignments to the DDNAMEs SYSIN, SYSPRINT, SYSOUT, SYSPUNCH, or SYSLIN. There are standard defaults for these ddnames (see Part II, "Input/Output Assignments" later). If an attempt is made to open a DDNAME of the form SYSUT_xxx_ (where _xxx_ are any three characters), *FAKEOS will create a sequential file with a size of ten pages. If the file already exists, *FAKEOS will empty it and use it. The files created will be named -FK#UT_xxx_, where the string _xxx_ is only the last three characters of the DDNAME. The second through sixth characters of these names may be specified by the user; he may also specify what is to be the default page size.

(1) The temporary file name option, which is given as

```
TMPNM=vvvvv
```

allows one to specify the five-character string used to build temporary file names. Temporary file names constructed for SYSUT_xxx_ DDNAMEs will be of the form _-vvvvvxxx_. The default value for _vvvvv_ is FK#UT.

(2) The temporary file size option, which is given as

```
TMPSIZ=n
```

allows one to change the default file size (in pages) for temporary files constructed as defaults when opening DDNAMEs of the form SYSUT_xxx_. If not specified, a default value of 10 pages is used.

(3) The no-prompting option, which is given as

```
NOPROMPT
```

disables any prompting for DDNAME-FDname assignments -- this is meaningful only in terminal mode.

Miscellaneous Options

(1) The input/output option, which is given as

```
IO
```

specifies that at *FAKEOS termination a summary is to be printed of all input/output for each DDNAME for which there was an OPEN request.

(2) The system-type option allows one to set the type of OS system. This is specified in the form

```
SYS=sss
```

where "sss" may be one of PCP, MFT, MVT, VS1, VS2, or CMS. When "sss" is CMS, the CVTRELNO field of the CVT is altered to contain the four-character string "CMS ". With any of the other options the appropriate bit pattern is placed at the CVT location CVTDCB (relative location X'0074' in the CVT). The setting of this byte affects how *FAKEOS services the following OS requests: WTL, GETMAIN, EXTRACT, IDENTIFY (will be an error for a non-PCP system, otherwise a NOP), and SNAP.

(3) The user may request that the program mask in the PSW be set to a given bit configuration at the time control is given to the initial object module. One specifies this as

```
SPM=h
```

where "h" is a hex digit.

(4) The user has control over the contents of the DCBDVTBL field in DCBs that are associated with files. This parameter also informs the *FAKEOS-BDAM module what the default direct-access device type is, so it can construct BDAM data sets correctly. In addition, this parameter will affect what *FAKEOS will return to the user program when a DEVTYPE macro with the DEVTAB or RPS option is issued. There is a parameter which sets the default for all of these and which can be specifically overridden with a modifier in the DDNAME-FDname assignment (see Part II). This parameter is specified as

```
D=dasd
```

where "dasd" must be one of 2301, 2302, 2303, 2305, 2311, 2314, or 3330 -- if not specified, the default is 2311.

(5) Certain messages from *FAKEOS may be suppressed with the option

```
TERSE=swtch
```

where "swtch" is either ON or OFF. If this option is not specified, *FAKEOS uses the global MTS TERSE switch. However, if the user program gets into trouble and enters ABEND processing, the effect of this option is nullified and all messages will appear. Use of this option makes it less obvious to the end-user that a program is running under *FAKEOS.

(6) Certain language processors allow the caller to specify a "heading line" for the printed output (see any of the IBM language processor programmer's guides about this--note how one would LINK to or ATTACH the associated language processor). To

cause *FAKEOS to specify a header in the call to the main program, the user should specify

```
     H=header
```

where "header" is a variable length string delimited by primes (internal primes must, as usual, be represented by two contiguous primes). When a header is specified, the parameter list passed to the initial object module consists of three words (rather than the normal one word). It is of the following form:

```
             DC  A(APARMS)
             DC  A(DDLST)
             DC  AL1(X'80'),AL3(HEADER)
```

where

```
     PARMS   DC  Y(j),CLj'normal-parm-string'
     DDLST   DC  Y(0)
     HEADER  DC  Y(k),CLk'header-string'
```

In this way one may pass two distinct parameter strings to the initial load. If one does not specify a header, then a one-word parameter list of the following form is generated:

```
             DC  AL1(X'80'),AL3(PARMS)
```

(7) The user may specify that he wants a trace to be made of his program. A trace entry is made each time that:

(a) an OS SVC is executed (code SVC),
(b) a program interrupt occurs (code PGNT),
(c) a READ/WRITE or physical I/O operation resulting from a GET/PUT occurs (code RDWT)--for a BDAM file the code is DARW.
(d) a CHECK macro is executed (code CHK)--for a BDAM file the code is BDCK,
(e) an STIMER exit occurs (code TIMR),
(f) a DCB exit is taken (code DCBE),
(g) a return is made from a LINKed module (code LNKR),
(h) a NOTE/POINT is issued (code NTPT),
(i) a control operation is issued using the CNTRL macro (code CNTL),
(j) a DCBEODAD exit occurs (code EOF),
(k) a SPIE exit is taken (code SPIX)
(l) a return is made from a SPIE exit routine (code SPIR).
(m) a DCBSYNAD exit is taken (code SYNE).

Each trace entry takes 76 bytes: the first 4 are the EBCDIC code as given above, the next 8 bytes contain the PSW at the time of the transfer of control (to *FAKEOS or to the exit routine), and the last 64 bytes reflect the contents of the general registers when control was transferred.

Traces may be made internally (in the user's virtual memory) or externally (onto a file). To specify an internal trace, the user should specify

```
T=n
```

where "n" indicates how many trace entries are to be maintained at any one time. A region of 76*n+16 bytes will then be allocated for this table. When the trace table is allocated, *FAKEOS will print a message indicating where it may be found. The first sixteen bytes of an in-core trace table contain control information. The first word contains the address of the next entry to be used, the second word contains the length of an entry, the third word contains the address of the last possible entry in the table, and the fourth word contains the address of the first possible entry in the table. When the table is full, wrap-around occurs.

To specify an external trace,

```
T=FDname
```

is used (the first character of FDname must not be numeric). After the program has run, the *FAKEOS utility in the file *FAKEOSTRACE may be used to print the trace file (SCARDS is assigned to the trace file).

(8) The model number field in the CVT (which is supposed to give the CPU model that OS is running on) can be set by specifying

```
MODEL=nn
```

where "nn" is the model number.

(9) The release number field in the CVT (which contains the release of OS) can be set by specifying

```
RELEASE=xxxx
```

where "xxxx" is the 1- to 4-character quantity to place in this field. The default is RELEASE=440.

Part II - Input/Output Assignments

The second part of the parameter list is optional. If one is using only the DDNAMES that are the same as MTS logical I/O unit names or SYSIN, SYSOUT, SYSPRINT, SYSPUNCH, SYSUTxxx, and/or SYSLIN, and the default specifications for these DDNAMEs are satisfactory, then this part may be omitted. The assignments in this part of the parameter list are characterized by specifications of the following form:

```
DDNAME=FDname[(b,e,i)][@modifiers][+...]
```

or

```
     DDNAME=U=LIOunit[@modifiers]
```

or

```
     DDNAME=UNIT=LIOunit[@modifiers]
```

where

DDNAME is the OS DDNAME,

FDname is an MTS FDname,

LIOunit is an MTS logical I/O unit name (e.g., SCARDS),

(b,e,i) is the line number range, where

```
   b   is the beginning line number
   e   is the ending line number
   i   is the line number increment
```

@modifiers is a list of modifiers each of which is:

```
   a legal MTS modifier (allowed with an FDname only)
   a record format modifier
   a valid DCB modifier
   a device type modifier
   a special processing modifier
```

Record format modifiers are as follows:

```
     @U{A|M}
     @F[B][S]{A|M}(blocksize[,recordsize])
     @V[B][S]{A|M}(blocksize[,recordsize])
     RECFM=any of the above
```

where

U is undefined record format (U format is not blocked)
F is fixed format
V is variable format
B indicates blocking
S indicates spanned records (VS and VBS only) or "standard" blocking (FBS) with all but the last being full size
A indicates standard ASA carriage-control (equivalent to @CC)
M indicates machine carriage-control characters (equivalent to @MCC)

For further information on record formats, see the section "Magnetic Tapes" in MTS Volume 19, Tapes and Floppy Disks.

The valid DCB modifiers are:

@DSORG={IS|PS|DA|PO} -- data set organization which is placed in the DCBDSORG field.

IS	= Indexed Sequential (not supported)
PS	= Physical Sequential (QSAM, BSAM, etc.)
DA	= Direct Access (BDAM)
PO	= Partitioned (BPAM)

@BUFNO=n -- "n" is a number between 0 and 255 inclusive which is placed in the DCBBUFNO field.

@BFTEK={S|A|R|E} -- buffer technique, which is placed in the DCBBFTEK field.

@BFALN={D|F} -- buffer alignment, which is placed in the DCBBFALN field.

@BUFL=n -- "n" is a number between 0 and 32767 inclusive which is placed in the DCBBUFL field.

@LIMIT=n -- "n" is a number which is placed in DCBLIMIT.

For further information on these parameters, see the IBM publications concerning OS data management.

The valid device type modifiers are of the form @D=dasd, where "dasd" is one of 2301, 2302, 2303, 2305, 2311, 2314, or 3330 and defaults to 2311. This is identical to the D=dasd parameter described earlier in the first parameter list, but applies only to the specified file.

The valid special processing modifiers are:

@DEBLOCK -- this specifies that a blocked output buffer is to be deblocked when writing on the associated FDname. Thus one could specify FB(4000,80)@DEBLOCK and write 80-byte lines to a file. DDNAMEs assigned to terminals or unit record devices are always deblocked.

@SYSIN|@SYSOUT -- this sets the SYSIN/SYSOUT flag in the Job File Control Block (JFCB) at JFCBTSDM. This flag will always be set for any DDNAME assigned to *MSINK*, *SINK*, *MSOURCE*, or *SOURCE*

@WRITE|@NOWRITE -- NOWRITE specifies that *FAKEOS is to allow an OPEN with the UPDAT modifier (implying some kind of output) even though the user really does not have write access to the file to which the DDNAME is attached. WRITE is the normal case and need not be specified. Normally FAKEOS will not allow one to OPEN a file with options which conflict with the MTS file access for that user with respect to the given file.

@REW|@NOREW -- NOREW specifies that the associated file is not to be rewound at CLOSE time no matter what were the options given with the CLOSE. Thus any further output (from a successive OPEN with the OUTPUT option) will be appended to the end of the file.

@PAD|@NOPAD -- will cause padding on input. For type-U records, the records will be padded with blanks to the blocksize given as "n" in @U(n) if needed. For type-V records, this modifier will cause truncated blocks to be padded to the length found in the block-descriptor word in the block.

@DISP={[NEW|OLD|MOD]|[PASS|KEEP|DELETE|CATLG|UNCATLG]} -- this will cause the DEBOPATB field in the DEB to be set. ICES examines this field, for example, to determine whether or not it should preformat a dataset.

@SPACE={n|(n,m)|(n,m,j)} -- where "n" is the number of "primary tracks" to be allocated, "m" is the secondary allocation quantity, and "j" is the number of PDS directory blocks to be allocated. At the moment, "j" is really ignored, and "n" and "m" are used to inform the *FAKEOS BDAM module how "large" (in the OS sense) a data set can be. This has nothing to do with the actual MTS size. The modifier corresponds to the SPACE keyword one can specify on a DD-card in OS.

TRACKS={n|(n,m)|(n,m,j)} -- this is identical to the SPACE modifier mentioned above.

Record format and DCB modifiers are processed as follows by *FAKEOS: Each of the modifiers corresponds to some field in both the DCB (Data Control Block) and the JFCB (Job File Control Block). There is exactly one JFCB for each DDNAME mentioned in the parameters to *FAKEOS which initially contains the values given in the modifiers specified. When a DCB is opened (by the OPEN SVC or macro), any field in it that has already been given a value (e.g., by the DCB macro) will be used unchanged. Any field in the JFCB that was not set by some modifier will be set based on the FDname that the DDNAME is assigned to. If this FDname is _not_ a magnetic tape, only the record format (U) and block size (obtained from GDINFO) will be set. If it is a magnetic tape, the record format, block size, record size, and many other fields in the JFCB will be set from the tape labels or CONTROL commands given previously.

Next, any DCB field which does not have a value will be set from the JFCB.

Whenever a DDNAME assigned to a magnetic tape is opened, the format, block size, record size, mode, and expiration date will be communicated to the MTS magnetic tape support routines which will record them in the tape labels, if any. Then, *FAKEOS will disable blocking by the tape support routines (unless @DEBLOCK was given, in which case it will be enabled) so that blocking is never done twice. All of these values will be restored when the DCB is closed if the tape is between data sets at that time.

When an attempt is made to open a file (via SVC OPEN), if the DDNAME is not found among those presented in the second part of the parameter list, a check is made to see if the DDNAME is one of the following: an MTS logical I/O unit which has been assigned, SYSIN, SYSPRINT, SYSOUT, SYSPUNCH, or SYSLIN. An MTS logical I/O unit will default to itself, SYSIN will default to SCARDS, SYSPRINT and SYSOUT will default to SPRINT, and SYSPUNCH will default to SPUNCH. If SCARDS, SPRINT, or SPUNCH are not specified, MTS will force the normal defaults. If SYSLIN is defaulted, it will be assigned to logical I/O unit 0 -- if logical I/O unit 0 is not specified, then *FAKEOS will create a temporary line file with the name -LOAD.

If an attempt is made to open a file with a DDNAME of the form SYSUTxxx and if this DDNAME is not found in the second part of the parameter list, then *FAKEOS will create a temporary sequential file the name of which is -FK#UTxxx. The size of this file is 10 pages. Both the default size and the first five characters of this filename may be set by user parameters (see the description of the first parameter list).

If a DDNAME has no default and was not specified in the parameters, *FAKEOS will prompt for an assignment if it is being run in conversational mode and NOPROMPT was not given. When replying to this prompt, one may enter the assignment for more than one DDNAME in the same format as Part II of the parameter list. If the first DDNAME assigned by this prompt is the one which caused it to occur, the "DDNAME=" may be omitted.

The following DDNAMEs have special significance to *FAKEOS. STEPLIBn and JOBLIBnn are assumed to be libraries from which programs can be loaded (as explained earlier in the section on resolving requests to LOAD, LINK, or XCTL). LINKLIB is assumed to contain BLDL information to be returned by a BLDL or FIND macro without a DCB parameter. WTLDD is the destination for output from the WTL macro if unit 17 is not assigned.

Partitioned Data Sets, FIND, and BLDL

A BLDL or FIND SVC or macro is processed by FAKEOS as follows: If no DCB parameter is present on the BLDL/FIND and the "BLDL=n" parameter was specified in parameter list 1, the information requested will be returned from the list constructed in virtual memory from the file assigned to LINKLIB. If neither the DCB parameter nor the "BLDL=n" parameter was present, the BLDL/FIND information will be read from the file assigned to LINKLIB; this must be a sequential file formatted as by *FAKEOSCREBLDL or a line file formatted as by *MACUTIL.

If a DCB parameter is specified on the BLDL/FIND, FAKEOS ensures that the DCB is DSORG=PO (partitioned) and is open. There are three possible assignments for the DDname associated with this DCB.

(1) If on a FIND, the DDname is SYSLIB and it is assigned to *DUMMY*, FAKEOS will attempt to find a file with the same name as the member name in the FIND call. If successful, the next read or write operation for SYSLIB will read or write this file. This feature is useful for translators such as COBOL which use SYSLIB for a source program library.
(2) If the DDname is assigned to a sequential file formatted as by *FAKEOSCREBLDL, the BLDL/FIND information will be returned from this file and, for a FIND, the read and write pointers for the file will be set to the member indicated. A $ENDFILE line terminates a member.
(3) If the DDname is assigned to a line file formatted as a macro library (see MTS Volume 14, _360/370 Assemblers in MTS_), the member name(s) will be located in the directory of the file. In this case, no "user data" will be returned, i.e., only the member name and TTR will be set in the BLDL output. No line number in a directory line may be greater than 16777. If the request is FIND, FAKEOS will set the line number so that the next read or write operation will read or write the member specified in the FIND. A $ENDFILE line terminates a member.

*FAKEOS Utility Programs

The program *FAKEOSCREBLDL is used to create a file to be used by *FAKEOS to service BLDL requests. The program is invoked by a command of the form

```
     $RUN *FAKEOSCREBLDL [logical I/O units]
```

where the logical I/O units are

```
     SCARDS - input BLDL information
     SPRINT - printed output
     SERCOM - error comments
     0      - BLDL data file for *FAKEOS (sequential file)
```

Each input line (read by SCARDS) should contain the BLDL information for one "member" of the simulated partitioned data set. The first 8 characters are the member name and the remainder of the line is the hexadecimal or binary representation of the BLDL information for the member. This may be the output produced on logical I/O unit 1 by *UNLINKER. The output file produced on logical unit 0 is usually attached to the DDNAME LINKLIB when running *FAKEOS.

The program *FAKEOSTRACE is used to print the trace file produced by the trace option (T=FDname). The program is invoked by a command of the form

```
     $RUN *FAKEOSTRACE [logical I/O units]
```

where the logical I/O units are

SCARDS - trace file
SPRINT - printed trace output

Examples:

```
$RUN *FAKEOS PAR=E=-LOAD+*COBLIB
```

The above example runs the COBOL object program in the file -LOAD.

```
$RUN *FAKEOS PAR=E=OBJFILE;;PARAMETER STRING FOR PROGRAM
```

The above example runs the COBOL object program in the file OBJFILE. The specified parameter string (Part III type) is passed to the program.

```
$RUN *COBOLU 0=OBJ1
$RUN *ASMG 0=*OSMAC SPUNCH=OBJ2
$RUN *FAKEOS PAR=E=OBJ1+OBJ2+*COBLIB;
   DDNAME1=FILE1@FB(800,80) DDNAME2=*SINK*
```

The above example compiles a COBOL program and an assembly language subroutine and executes them as a single program. This program refers to two DDnames - DDNAME1 and DDNAME2 - in addition to the names defaulted by *FAKEOS.

*FILEDUMP

Contents:	The file-dumping program.
Purpose:	To dump the contents of a file or device in hexadecimal format. Note: *TAPEDUMP may also be used to dump files, and is generally preferable.
Use:	The file-dumping program is invoked by the $RUN command.
Program Key:	*FILEDUMP

Logical I/O Units Referenced:

SCARDS	the file to be dumped.
SPRINT	the dumped contents of the file.
GUSER	the list of file names if SCARDS is not assigned and the file name was not given in the PAR field. An end-of-file will terminate the list of files.
SERCOM	prompting messages and error comments.

Parameters: The file name may be specified in the PAR field of the $RUN command. The output on SPRINT is normally 64 columns per line; the user may vary the length of the output line with the LENGTH parameter. The LENGTH may vary between 1 and 132 inclusive. A blank must separate the file name from the LENGTH parameter.

Examples:

```
$RUN *FILEDUMP  SCARDS=MYOBJ
```

In the above example, the file MYOBJ is dumped.

```
$RUN *FILEDUMP SCARDS=-OBJ SPRINT=-PRINT PAR=LENGTH=128
```

In the above example, the file -OBJ is dumped into the file -PRINT with an output line length of 128 bytes.

```
$RUN *FILEDUMP SPRINT=-PRINT PAR=-OBJ LENGTH=128
```

The above example is equivalent to the previous example.

Sample Terminal Run:

The input entered by the user is in lowercase; system output is in uppercase.

```
#$run *filedump
#EXECUTION BEGINS

 ENTER NAME OF FILE TO DUMP:
 -file(1,2)
 LINE NO.       1    ;  80 CHARACTERS
 02C5E2C4404040404040003040400001C4E4D4D7C640404000000000400007DC
 E2C5D9C3D6D4404002000000404040E2C3C1D9C4E2404002000000040404040
 40404040404040C4D4D7C6F0F0F0F1
 LINE NO.       2    ;  80 CHARACTERS
 02C5E2C4404040404040003040400004C7C5E3C6C4404040020000000404040
 D9C5C1C440404040400200000040404040E2D7D9C9D5E34040020000000404040
 40404040404040C4D4D7C6F0F0F0F2
 ENTER NAME OF FILE TO DUMP:
 $endfile
#EXECUTION TERMINATED
```

*FILESCAN

Contents: The file-scan program.

Purpose: To locate a line in a file according to a given format and print the line number and the contents of the line.

Use: The program is invoked by the $RUN command.

Program Key: *FILESCAN

Logical I/O Units Referenced:

```
                SCARDS - file to be scanned, if assigned.
                SPRINT - line number and contents of selected line.
                GUSER  - format and file-name input.
                SERCOM - requests and error comments.
```

Description: A format may be entered via the PAR field on the $RUN command or in response to a request by the program. If the PAR field is used, execution will terminate after processing the single format. If the PAR field is not used, the program will prompt via SERCOM for the format. There are three types of format terms: skip, transfer, and character string. These are entered via GUSER. The skip term consists of the letter "S", followed by an optional minus sign, followed by a string of decimal digits. The value of the digit string is the number of columns to be skipped. The transfer term consists of the letter "T" followed by a string of decimal digits. The value of the digit string is the column where the next skip or character string term will start. The character string term is delimited by primes. The characters in the string are compared to the appropriate columns of the lines of the file to be scanned. A maximum of 20 character strings may be used in one format. If a prime is desired in the character string, it may be represented by two consecutive primes. (Note: Unless it is the last character in a character string, each prime reduces the maximum number of character strings by one.) Commas may be used to separate format terms, but they are not mandatory. The format is terminated by the first blank not in a character string or by the end of the line. If the first character entered is a ">", it is assumed that the characters immediately following name a file and that the previous format is to be used. If "CONTINUE" (or "C") is entered, the scan is continued with the next line in the same file using the same format. If "ALL" (or "A") is entered, the scan is continued with the next line in the same file using the same format; and all lines in the remaining part of the file that match the format are printed.

The prime delimiting the beginning of a character string in a format may be immediately preceded by one of the following character combinations:

```
     <  ¬  >  ¬<  ¬>
```

In this case, the corresponding character string in the file line must be less-than, not-equal-to, greater-than, not-less-than, or not-greater-than the format character string. A minus-sign may be substituted for each occurrence of the not-sign if desired. When more than one character string is to be compared, the relationship between them is a "logical and". That is,

```
     T10,<'OMEGA',S-5,>'ALPHA'
```

would locate a line where the five-character string starting in column 10 is greater than "ALPHA" and less than "OMEGA" according to the IBM 360/370 collating sequence. Since ">" indicates a file name when it occurs in column 1, it is necessary to use some other character in column 1 of a format. Therefore, if it is desired to locate a line with a character string starting in column 1 which is greater than "EXAMPLE", one of the following formats could be used:

```
     T1,>'EXAMPLE'
     S0,>'EXAMPLE'
```

After the format is entered, the program will scan the file assigned to SCARDS. If SCARDS is not assigned, then the program will prompt for the file via SERCOM. The file name is entered via GUSER and may be followed by a line number range and/or modifiers. The first blank or end of line terminates the file name. Lines in the file may be any length. If "CONTINUE" (or "C") is entered, the scan is continued with the next line in the same file.

An end-of-file when a file name is requested will cause a format request. An end-of-file when a format is requested will cause termination.

Example: `$RUN *FILESCAN SCARDS=FILE1 PAR=T10'LA',S4'R4,'`

The above example will scan the file FILE1 for lines containing "LA" in columns 10-11 and "R4," in columns 16-18.

*FSAVE

Contents: The file-save and restore program.

Purpose: To save and restore files (sequential or line) on magnetic tape.

Program Key: *FSAVE

Use: The program is invoked by the $RUN command.

Logical I/O Units Referenced:

```
SCARDS - the source of instructions to *FSAVE.
SPRINT - program messages and error comments.
SERCOM - error messages
GUSER  - prompting, if in conversational mode.
0      - the  pseudodevice name of the 9-track tape to be
         used.
```

Description: *FSAVE reads instructions from SCARDS, prints messages and error comments on SPRINT, and prompts on GUSER and SERCOM.

The lines from SCARDS contain commands (followed by the filenames which the commands are to operate on in the case of SAVE and RESTORE). The commands must be preceded by three dots (...) to distinguish them from file names. One letter abbreviations of the commands are allowed.

The commands are:

...SAVE

The SAVE command saves the files whose names whose names are given on lines following the "..SAVE" command. If the line contains one name, the file by that name is saved on the tape and remembered on tape by the same name. If the line contains two names, the first name is taken as the file name on disk but the file is labeled on tape with the second name. The first name (or pair of names) may be on the same line as ...SAVE.

Do not specify a line number range on any of the file names. *FSAVE will save the entire contents of the file (from line -99999.999 to +99999.999). Each line will retain its original number, if restored to a line file.

...RESTORE

The RESTORE command restores the files whose names are given on lines following the "...RESTORE" command. If the line contains one name, the file on tape is restored to a file on disk by the same name. The first name (or pair of names) may be on the same line as ...RESTORE.

If the line contains two names, the first name is the name of the file on tape and the second is the name of the disk file into which it is to be restored. *FSAVE empties the disk file before restoring. If ...RESTORE is followed by ...ALL, all files on the tape will be restored. If the disk file is not the same type as the file originally saved, this will be noted on SPRINT and the user will be given the opportunity to cancel the request in terminal mode.

File names are remembered as they are entered and all restore operations are done when the next command (for example, ...END) is entered.

...LIST

The LIST command lists all the files currently on the tape.

...END

The END command terminates the ...SAVE or ...RESTORE operation.

Notes: *FSAVE maintains a table of contents at the end of the tape (it has to be at the end so that the tape can have data added to it --- a record at the beginning of the tape cannot be safely overwritten). First, *FSAVE skips to the end of the tape and reads the table of contents. At this point, the tape is positioned correctly for saving additional files. If several files are to be saved and restored in the same run, it is more efficient to save first, then restore, since restore requires that the tape be rewound. The LIST command works from the in-core copy of the table of contents and can be executed at any time without penalty.

The first time a tape is used, PAR=INIT must be specified so that *FSAVE does not attempt to find a nonexistent table of contents. When the same tape is later used, PAR=INIT must not be specified since this would cause *FSAVE to overwrite files that have been previously saved.

The maximum number of files that may be saved on a tape is 250.

The maximum number of letters in a file name is 16.

*FSAVE uses "%" for a prefix character when run from a terminal.

The tape used may be either labeled or unlabeled. If it is labeled, the format will be U(32767) and the data-set name for each file saved will be the same as the name in the table of contents for that file, except that "*" and "," will be changed to "?".

The user should not interrupt *FSAVE or disconnect the terminal when the last operation was a save of a file. This will result in a tape which can not be read by *FSAVE since it has no table of contents. It is possible to reconstruct a table of contents that has been lost for one reason or another. It should consist of one file (dataset, if the tape is labeled) containing one 4004-byte record. The first 4000 bytes consist of 250 16-byte entries, each of which is the name of one file on the tape. These should be in the same order as the files on the tape (the order given by ...LIST). The last 4 bytes contain a fullword binary integer giving the number of bytes actually used in the first part, i.e., the number of files on the tape times 16. A directory of this form will indicate that all files are line files and no files are deleted. The table of contents should be followed by two tape-marks (a null data-set if the tape is labeled). For assistance in reconstructing a table of contents, contact the Computing Center.

Parameters: PAR=INIT must be specified the first time that any files are to be saved on the tape.

Examples:

```
$MOUNT PAR=C9999 9TP *TAPE* RING=IN 'TAPE ID'
$RUN *FSAVE 0=*TAPE* PAR=INIT
...SAVE
X
Y FILE2
Z
$ENDFILE
```

This example initializes a tape and saves the files X, Y, and Z. The file Y is saved under the name FILE2.

```
$MOUNT PAR=C9999 9TP *SAVE* RING=IN 'TAPE ID'
$RUN *FSAVE 0=*SAVE*
...SAVE Q
FORT SOURCE
...RESTORE
RFILE
OBJ2 XRQ
...LIST
$ENDFILE
```

This example uses a tape that previously has been used for a file save. It adds the files Q and FORT (under the name SOURCE) to the tape, restores the files RFILE and OBJ2 (OBJ2 is restored to the disk file called XRQ), and lists the names of all the files on the tape.

*FSCON

Contents: The *FSAVE-to-*FS conversion program.

Purpose: To convert files from a tape created by *FSAVE to a format acceptable to the *FS program.

Use: The program is invoked by a $RUN command.

Program Key: *FSCON

Logical I/O Units Referenced:

GUSER	- the source of commands to *FSCON (defaults to *MSOURCE*).
SPRINT	- listings of the directories of files on the tapes (defaults to *SINK*).
SERCOM	- error messages and program comments (defaults to *MSINK*).
0	- the pseudodevice name of the magnetic tape that was created by *FSAVE.
1	- the pseudodevice name of the magnetic tape to which the converted files will be written. This must be a 9-track tape mounted with RING=IN.

Parameters: One of the following two parameters may be specified in the PAR field on the $RUN command. RECOVER is the default parameter.

INIT The INIT parameter must be specified when the tape attached to logical I/O unit 1 is used for the first time. The tape will be rewound and two tapemarks will be written, indicating the tape is empty.

RECOVER The RECOVER parameter may be specified on any subsequent use of the tape. Since, with this option, the tape is assumed to contain saved files, files that are converted will be written after those that already exist on the tape.

Commands: The following commands are available for *FSCON. The underlined portion of each command name may be used as an abbreviation.

CONVERT oldname [newname]

This command specifies that the file "oldname" on logical I/O unit 0 is to be converted and written on logical I/O unit 1 using "newname" as the name of the file. If "newname" is omitted, "oldname" will be used as the name of the file when written.

```
E.g., CONVERT TEST
      C TEST PROG
```

<u>C</u>ONVERT (firstfile#) [(lastfile#)]

This command specifies that successive files starting with the file numbered "firstfile#" and ending with the file numbered "lastfile#" on logical I/O unit 0 are to be converted and written on logical I/O unit 1. "firstfile#" and "lastfile#" are as obtained from the directory for I/O unit 0. "lastfile#" may be omitted, in which case, only the file numbered "firstfile#" will be converted and written.

```
E.g., CONVERT (7) (11)
```

<u>L</u>IST0

The directory of files currently stored on the tape attached to logical I/O unit 0 is printed.

<u>LIST1</u>

The directory of files currently stored on the tape attached to logical I/O unit 1 is printed.

<u>S</u>TOP

All queued conversion requests are processed; then execution is terminated. In batch mode, an $ENDFILE in the command stream is identical to a STOP command. In conversational mode, entering $ENDFILE causes all queued conversion requests to be processed; the user is then prompted about continuing execution.

Description: In order to minimize tape movement, all CONVERT requests are queued. This means, in a particular conversion queue, all files converted will maintain the same relative order as they had on the *FSAVE tape.

Examples: An initial run with *FSCON. The files numbered 1, 2, and 4 on the *FSAVE tape are to be converted and written on the *FS tape.

```
$MOUNT
C0000 9TP *IN* 'FSAVE TAPE'
C0001 9TP *OUT* RING=IN 'FS TAPE'
$ENDFILE
$RUN *FSCON 0=*IN* 1=*OUT* PAR=INIT
```

```
CONVERT (1) (2)
CONVERT (4)
STOP
```

A subsequent run with *FSCON. The files named DATAFILE1 and DATAFILE2 on the *FSAVE tape are converted and written on the end of the *FS tape.

```
$MOUNT
C0000 9TP *IN* 'FSAVE TAPE'
C0001 9TP *OUT* RING=IN 'FS TAPE'
$ENDFILE
$RUN *FSCON 0=*IN* 1=*OUT*
CONVERT DATAFILE1 DATACOPY
CONVERT DATAFILE2
LIST1
STOP
```

*GENLIB

Contents: The library-generating program.

Purpose: To generate a loader library from object modules.

Use: The program is invoked by the $RUN command.

Program Key: *GENLIB

Logical I/O Units Referenced:

SCARDS	either the input file containing the object modules to be added to a library or a sequence of commands.
SPRINT	printed output produced by the library generator.
SPUNCH	default unit for a library of object modules added by the library generator.
SERCOM	diagnostic messages.
GUSER	user responses in conversational mode.

Description: Those users who want only to generate a library of object modules can simply issue the following command:

```
$RUN *GENLIB SCARDS=inFDname SPUNCH=outFDname
```

In the above case, object modules are read from "inFD-name", converted to the optimized format, and written on "outFD name". The logical structure of the modules is completely preserved with the following exceptions:

(1) The duplicate modules and modules with only blank names 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) COM records are discarded.
(5) Records other than SYM, ESD, TXT, CSI, RLD, END, REP, and nonabsolute DEF, are placed in front of the output modules. An example is a NCA record.

The PAR field allows the user some control over the processing done by the library generator. Those users who wish to use the more advanced features of the library generator must use the library-generator commands as described below. Commands are read from SCARDS and printed output is written on SPRINT; thus the typical

$RUN command to use the library generator in command mode is:

```
     $RUN *GENLIB
```

Command input is terminated by an end-of-file, by a RETURN command, or by a STOP command. The MTS logical I/O unit SPUNCH must not be specified.

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 "-", "¬", "NO", or "N".

<u>COM</u>SAVE	COMSAVE specifies that COM (comment) records are to be saved during processing. NOCOMSAVE specifies that COM records are not saved. The default is NOCOMSAVE.
<u>EMP</u>TY	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.
<u>FILL</u>=hh	Two hexadecimal digits "hh" specifies a character to fill the gaps. The default is FILL=81.
<u>GAP</u>SIZE=n	"n" specifies the maximum object module text gap size to be filled during the library generator processing. The default is ORL/2 or 256, whichever is smaller.
<u>MSG</u>SAVE	MSGSAVE specifies that MSG (message) records are to be saved during processing. NOMSGSAVE specifies that MSG records are not saved. The default is MSGSAVE.
<u>OPT</u>	OPT specifies that object modules are to be optimized, i.e., reformatted according to ORL, FILL, and GAPSIZE. NOOPT specifies that input records of object modules are to be copied as is. The default is OPT.
<u>ORL</u>=n	"n" specifies the maximum output record length to be used for output produced by the library generator. The default is 32767.
<u>SLOTS</u>=n	"n" specifies the number of slots to be allocated for a directory record of a library. The default is 128 for sequential

	files, or minimum (the least possible number of slots) for line files.
SORT	SORT specifies that the slots in a directory record are to be sorted by name. NOSORT specifies that the slots are sorted by their line numbers or sequential pointers. The default is NOSORT.
SYMSAVE	SYMSAVE specifies that SYM (symbol) records are to be saved during processing. NOSYMSAVE specifies that SYM records are not saved. The default is SYMSAVE.
VERIFY	VERIFY specifies that verification for each command is requested. NOVERIFY suppresses the verification. The default is VERIFY.
XREF	XREF specifies that the cross-reference listing is to be printed (only if both SCARDS and SPUNCH are specified).

The complete description of the library-generator command language is given on the following pages.

Library-Generator Command Language

The general form of a command is:

```
     commandname[@modifier]...[ operand]...
```

Modifiers may be prefixed by "¬", "-", "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 is the syntax notation for the library-generator 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 alternates.
____	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 use:

(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 an MTS continuation character (currently 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. 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 library generator.
(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 library-generator commands and their applicable modifiers.

Command	Operand	Applicable Modifiers
ADD	[FROM] FDname [slist]	COMSAVE, GAPSIZE=, MSGSAVE, OPT, ORL=, SYMSAVE, VERIFY
CLEAR	None	VERIFY
COMMENT	comment	None
CREATE	FDname [lhs=rhs[,]lhs=rhs...]	VERIFY
DELETE	slist	DIR, VERIFY
EMPTY	FDname	VERIFY
EXPLAIN	[ON FDname] [list]	None
INCLUDE	[FROM] FDname [slist]	COMSAVE, GAPSIZE=, MSGSAVE, OPT, ORL=, SYMSAVE, VERIFY
LIST	[ON FDname] [object [[FOR] slist]]	None
MAP	[ON FDname] [slist]	FULL
MCMD	MTS command	None
MTS	[optional command]	None
PUNCH	[ON] FDname [slist]	EMPTY, GAPSIZE=, OPT, ORL=, VERIFY
RENAME	old1[=]new1[[,]old2[=]new2]]...	VERIFY
REPLACE	[FROM] FDname [slist]	COMSAVE, GAPSIZE=, MSGSAVE, OPT, ORL=, SYMSAVE, VERIFY
RETURN	None	None
SET	lhs=rhs [[,] lhs=rhs]...	None
SNIFF	None	None
STOP	None	None
XREF	[ON FDname] [slist]	None

Library Generator Commands

Command: ADD [FROM] FDname [slist]

Modifiers: COMSAVE, GAPSIZE, MSGSAVE, OPT, ORL, SYMSAVE, VERIFY

Example: `ADD FILE1+FILE2(100,199) ALLBUT QQ`

Explanation: Modules are read in from the specified file or devices and then added to the current library. COM records are discarded unless the COMSAVE modifier is specified.

Command: CLEAR

Modifiers: VERIFY

Example: `CLEAR@NV`

Explanation: The CLEAR command finishes the current library if any, and completely clears out the internal structure.

Command: COMMENT comment

Modifiers: None

Example: `COMMENT - Now we delete some symbols.`

Explanation: The COMMENT command is useful for documenting sequences of commands given to the library generator. Note that command lines beginning with an asterisk "*" are also treated as comments.

Command: CREATE filename [lhs=rhs[[,]lhs=rhs...]]

Modifiers: VERIFY

Example: `CREATE SEQLIB TYPE=SEQ SLOTS=50 SIZE=10P`

Explanation: This creates a library file. The filename is acquired unless it already exists. The optional keywords are:

```
SIZE=nP
MAXSIZE=nP
TYPE={LINE|SEQ}
VOLUME=volname
SLOTS=n
```

The keywords are the same as those for the MTS command $CREATE except that SIZE and MAXSIZE are expressed in terms of pages and SLOTS specifies the number of the external symbol entries for a library directory. The default for SLOTS is 128 for sequential files and the minimum (least possible number of slots) for line files. SLOTS must be in a range between 1 and 2730. A sufficient number of SLOTS should be allocated for sequential files so that the library generator can write out a directory record before writing the first module.

Command:	DELETE slist
Modifiers:	DIR, VERIFY
Example:	DELETE SUBRA, SUBRB
Explanation:	This command removes the specified external symbols from the directory. If DIR modifier is specified, only the symbols are removed, and a module is removed when it has all symbols deleted. Otherwise, modules defining the specified symbols to be deleted are removed.

Command:	EMPTY filename
Modifiers:	VERIFY
Example:	EMPTY A
Explanation:	The specified filename is emptied and then becomes the current library file.

Command:	EXPLAIN [ON FDname] [list]
Modifiers:	None
Example:	EXPLAIN ON *PRINT* EVERYTHING
Explanation:	This command explains the list on SPRINT or on FDname, if specified. Applicable symbols of the list are:

$, *, ?, ADD, CLEAR, commands, COMMENT, CREATE, DELETE, description, EMPTY, example, EXPLAIN, everything, INCLUDE, LIST, MAP, MCMD, modifiers, MTS, parameters, PUNCH, RENAME, REPLACE, RETURN, SET, SNIFF, STOP, XREF, @COMSAVE, @DIR, @EMPTY, @FULL, @GAPSIZE, @MSGSAVE, @OPT, @ORL, @SYMSAVE, @VERIFY.

Command:	INCLUDE [FROM] FDname [slist]
Modifiers:	COMSAVE, GAPSIZE, MSGSAVE, OPT, ORL, SYMSAVE, VERIFY
Example:	INCLUDE FILE1+FILE2(100,199) ALLBUT QQ
Explanation:	Modules are read in from the specified file or device and then included to the current library. COM records are discarded unless the COMSAVE modifier is specified.
Note:	This is synonymous to the ADD command.

Command:	LIST [ON FDname] [object [[FOR] slist]]
Modifiers:	None
Example:	LIST ON *PRINT*
Explanation:	This command lists all modules and entry points. "slist" may be used to list only selected modules. The listing is normally printed on SPRINT unless "ON FDname" is given. "object" may be any one of the following:

OMS	- list all module names with their corresponding line numbers or sequential pointers
ENTRYS	- list all defined symbols (types SD, LD, LR) in the directory
LDS	- list all type LD entry point symbols
LRS	- list all type LR entry point symbols
CSECTS	- list all control section names
PCS	- list modules containing private control sections
PRS	- list all pseudo-register definitions
ERS	- 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 library

Command:	MAP [ON FDname] [slist]
Modifiers:	FULL
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 pseudo-registers, is printed. The MAP listing is printed on the SPRINT output unless "ON FDname" is given.

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 library generator. Alternatively, the user may issue the MTS command directly in library generator command mode by prefixing it with a dollar sign, e.g., $EMPTY -OBJ. Warning: the program will not know if the user changes the library.

Command:	MTS [MTS command]
Modifiers:	None
Example:	MTS EMPTY OBJECT OK MTS
Explanation:	If the MTS command is specified, it is executed by MTS, and the library generator may be reentered with a $RESTART MTS command.
Note:	The current library will not be completed unless the user issues one of the commands: CLEAR, CREATE, EMPTY, SET LIBRARY, STOP, and RETURN. Also the program will not know if the user changes the library. Alternatively, the user may issue just the MTS command. It will then turn to MTS mode; the library generator may be entered by $RESTART command.

Command: PUNCH [ON] FDname [slist]

Modifiers: EMPTY, GAPSIZE, OPT, ORL, VERIFY

Example: PUNCH ON -X ONLY SORT

Explanation: If no "slist" is given, all modules of the current library is punched on FDname. Otherwise, modules specified will be punched. In addition, if the EMPTY modifier is specified, the FDname is first emptied before the punching.

Command: RENAME old1[=]new1 [[,]old2[=]new2] ...

Modifiers: VERIFY

Example: RENAME LIBGEN=GENLIB

Explanation: The RENAME command causes one or more specified symbols to be renamed. Only control sections and label definitions can be renamed. If a module contains SYM records, relevant symbols will not be renamed. Equal signs may be omitted.

Command: REPLACE [FROM] FDname [slist]

Modifiers: COMSAVE, GAPSIZE, MSGSAVE, OPT, ORL, SYMSAVE, VERIFY

Example: REPLACE FROM -NEWLIB

Explanation: This is same as the ADD or INCLUDE commands except if modules in FDname exist in the current library, they are replaced.

Command: RETURN

Modifiers: None

Example: RETURN

Explanation: The library generator completes the current library if any, and terminates the processing. This is identical to the STOP command. An end-of-file in the command stream also terminates the processing.

Command:	SET lhs=rhs[[,]lhs=rhs]...
Modifiers:	None

Examples:	SET LIBRARY=MYLIB SLOTS=25
	SET (COMSAVE,MSGSAVE)=ON

Explanation:	The commands sets the global parts. Locally the user may set them by using the command modifiers. Available keywords are:

COMSAVE={ON|OFF}	Defaults to OFF
ECHO={ON|OFF}	ON if batch, OFF otherwise
ENDJUNK=name	Defaults to "ENDJUNK"
FILL=hh	Defaults to X'81'
GAPSIZE=nnn	Defaults to min(ORL/2,256)
LIBRARY=FDname	Defaults to SPUNCH or none
MODCHAR=character	Defaults to "@"
MSGSAVE={ON|OFF}	Defaults to ON
OPT={ON|OFF}	Defaults to ON
ORL=nnn	Defaults to 32767
SLOTS=nn	Defaults to 128 for sequential file, minimum for line file
SORT={ON|OFF}	Defaults to OFF
SYMSAVE={ON|OFF}	Defaults to ON
VERIFY={ON|OFF}	Defaults to ON

Command:	SNIFF
Modifiers:	None
Example:	SNIFF
Explanation:	This command lists the filename, its file type, number of modules, number of entries, and number of free slots.

Command:	STOP
Modifiers:	None
Example:	STOP
Explanation:	The library generator completes the current library if any, and terminates the processing. This is identical to the RETURN command. An end-of-file in the command stream also terminates the processing.

Command: XREF [ON FDname] [slist]

Modifiers: None

Example: XREF

Explanation: For each module, all external symbols "referenced" from that module are printed. A "reference" may refer to another control section, a common section. The cross- reference listing is printed out in two forms: one showing modules with all their references, and the other showing references with all modules referenced. If "slist" is specified, only modules in "slist" will have their references shown. 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 modifiers are prefixed by "@" or MODCHAR as set by the SET MODCHAR command (see the SET command) and appended to the commands. The modifiers may be negated by applying "¬", "-", "NO", "N".

Modifier: @COMSAVE

Example: ADD@COM MYFILE

Explanation: This modifier preserves COM records in ADD, INCLUDE, REPLACE, and PUNCH commands. The default is @¬COM-SAVE, i.e., "throw away" COM records. See also the command description for SET COMSAVE=ON.

Modifier: @DIR

Example: DELETE@DIR Z

Explanation: The DIR modifier may be applied to the DELETE command to specify that the operands of the command are to be removed from the directory (as opposed to object modules).

Modifier: @EMPTY

Example: PUNCH@EMPTY ON SORTFILE ONLY SORT

Explanation: This modifier may be applied to the PUNCH command to request that the output file is to be emptied before output from the library generator is written to it.

Modifier: @FULL

Example: MAP@FULL

Explanation: This modifier may be applied to the MAP command to specify that the entire external symbol dictionary is to be printed.

Modifier: @GAPSIZE=n

Example: ADD@GAPSIZE=400 SEQFILE

Explanation: When executing commands such as ADD, INCLUDE, PUNCH, and REPLACE, the library generator will fill in holes of size <= GAPSIZE (for which no text was received) with a fill character, which defaults to X'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: @MSGSAVE

Example: ADD@-MSG USERS

Explanation: This modifier preserves MSG (message) records in ADD, INCLUDE, PUNCH, and REPLACE commands. By default, MSG records are preserved. See also the SET command description for SET MSGSAVE=ON.

Modifier: @OPT

Example: REPLACE@¬OPT FILE ALLBUT IHENTRY

Explanation: Object modules are normally optimized according to ORL, GAPSIZE, and FILL. No optimization forces the library generator to copy object modules with no conversion. See also the SET command description for SET OPT=ON. By default, the object modules are optimized.

Modifier: @ORL=n

Example: PUNCH@ORL=80 LINEFILE ONLY LAND

Explanation:	ORL stands for output record length and defines the maximum record size the library generator will write for ADD, INCLUDE, PUNCH, REPLACE commands. The default for ORL is 32767.

Modifier:	@SYMSAVE
Example:	INCLUDE@SYM MYFILE
Explanation:	SYMSAVE preserves SYM records in ADD, INCLUDE, PUNCH, and REPLACE commands. By default, SYM records are preserved. See also the SET command description for SET SYMSAVE=ON.

Modifier:	@VERIFY
Example:	ADD@V -LOAD
Explanation:	If verification has been disabled globally via SET VERIFY=OFF, then it can be enabled for a particular command via the VERIFY modifier.

Example

```
#$run *genlib
#EXECUTION BEGINS

    LIBRARY GENERATOR -- VERSION (FEB 2, 1977)
*set library=*library
    *** WARNING--LIBRARY FILE CANNOT BE USED AS OUTPUT
   SEQF FILE "*LIBRARY" HAS 54 MODULES, 140 ENTRY POINTS,
             0 FREE SLOTS
*map@f spie

   ESID SYMBOL   TP ADDR   LTH    AF

   0001 SPIE     SD 000000 000170
   0002 PGNTTRP  ER
   SIZE=000170
*list ers sort
 SORT    -CASECONV CHGFSZ   CHKFDUB  CLOSEFIL CNFGINFO CONTROL
          CREATE   CUINFO   DESTROY  EMPTY    FREEFD   FREESPAC
          GDINFO   GDINFO2  GETFD    GETSPACE GUINFO   GUSER
          LINK     LOAD     QUIT     READ     SERCOM   SETLIO
          SORTE0   SORTE1   SORTE2   SORTE3   SORTE4   SORTE5
          SORTE6   SORTE7   SORTE8   SORTE9   SYSTEM   TRUNC
          WRITE
*list
 $SPACE   001A00B2 #FPCON   00010E98 @TESTITP 000106E2
 ARINIT   001900C8 ATNTRP   001303A3 BTD      00130DA3
 DTB      00150363 DUMP     00180B09 EQUC     00140A7D
 FINDC    00140B25 FINDST   00140D85 FREAD    00150517
 GDINF    001304F3 GRAND    001A0754 GRJLDT   000A0EFB
 GRJLTM   000A0FF5 GROSDT   000B0303 GTDJMS   00130A8B
 GTDJMSR
 ATTN !!
*punch@orl=80 on -line only land
 PUNCHED: IHCLOGIC SHFTR    SHFTL    LCOMPL   COMPL    LXOR
 XOR      LOR      OR       LAND     AND
*list endjunk

  THE FOLLOWING IS THE LISTING OF ENDJUNK
  RIP  <EFL>
  RIP  <FIX>
  LCS           LCSYMBOL
  LCS           <EFL>
  LCS           <FIX>
  LDT
*punch on -junk endjunk
 PUNCHED: ENDJUNK
*create -library
 FILE "-LIBRARY" HAS BEEN CREATED
*add *time
 ADDED:   TIME     ENDJUNK
*add *objscan
 ADDED:   OBJSCAN
*rename objscan scan
 OLD: "OBJSCAN" NEW: "SCAN"
*add *objlist
 ADDED:   INTERP   SYMINTRP
*delete@dir symintrp
 DELETED: SYMINTRP
*sniff
 LINE FILE "-LIBRARY" HAS 3 MODULES, 3 ENTRY POINTS
*list
 INTERP       15   SCAN          8   TIME          3
*replace@orl=256@gap=80 *objlist
 REPLACED:         INTERP   SYMINTRP ENDJUNK
*add *users
 ADDED:   USERS
*list misc users
 USERS   - MSG  The MTS $SYSTEMSTATUS command may be used to
           MSG  obtain this same information.
*delete intrp
 *** SYMBOL "INTRP" DOES NOT EXIST
 DELETED:
*delete interp
 DELETED: INTERP   SYMINTRP
*list
 SCAN          8   TIME          3   USERS       109
*list entrys
```

```
  MODULE - ENTRY POINT NAMES
 SCAN    -SCAN
 TIME    -TIME
 USERS   -USERS
*xref time users

  MODULE - EXTERNAL REFERENCES

 TIME    -FREESPAC GDINFO   GUINFO   MSG      SERCOM   SPRINT
 USERS   -JOBSTA   JTBLLIM  SPRINT   USERSCON

  SYMBOL - REFERENCED BY MODULES

 FREESPAC-TIME
 GDINFO  -TIME
 GUINFO  -TIME
 JOBSTA  -USERS
 JTBLLIM -USERS
 MSG     -TIME
 SERCOM  -TIME
 SPRINT  -TIME     USERS
 USERSCON-USERS
*add *pl1lib only pl1fcn
  *** WARNING: COM RECORDS ARE BEING IGNORED
 ADDED:   PL1BEG   LG1FCN   LGLFCN   CDFCN    CPXFCN   I2FCN
 IPLFCN   PLDFCN   PL1FCN   PL1SUB   PL1END   IPL1RC   IHEMAIN
*delte@dir ihemain
 WARNING: ASSUMING "DELETE" FOR "DELTE"
 DELETED: IHEMAIN
*add *pl1lib ihemain
  *** WARNING: COM RECORDS ARE BEING IGNORED
 ADDED:   IHEMAIN
*list
 IHEMAIN     123   PL1BEG      116   SCAN          8
 TIME          3   USERS       109
*list entrys

  MODULE - ENTRY POINT NAMES
 IHEMAIN -IHEMAIN
 PL1BEG  -CDFCN    CPXFCN   IPLFCN   IPL1RC   I2FCN    LGLFCN
          LG1FCN   PLDFCN   PL1BEG   PL1END   PL1FCN   PL1SUB
 SCAN    -SCAN
 TIME    -TIME
 USERS   -USERS
*stop

  THE FOLLOWING CARDS ARE ADDED
 LCS           LCSYMBOL
 LDT
 LINE FILE "-LIBRARY" HAS 5 MODULES, 16 ENTRY POINTS

 CPU TIME        .91 SECS
#EXECUTION TERMINATED
```

*KDFLIB

Contents: The English Electric KDF9 ALGOL output procedure library for programs compiled by *ALGOL.

Use: *KDFLIB should be specified with the CODE parameter in the PAR field of the $RUN command, i.e.,

```
     $RUN object+*ALGOLLIB ...  PAR=CODE=*KDFLIB
```

Program Key: *EXEC

Description: The KDF9 I/O procedures are stored as external procedures in *KDFLIB. Therefore, each procedure used must be declared in the outer block of the program. These provide facilities for simple formatted output not available in the standard OS/ALGOL. A program using the WRITE procedure might start:

```
     'BEGIN'
     'PROCEDURE' WRITE(DSN,FE,AE);
     'VALUE' DSN,FE,AE;
     'INTEGER' DSN,FE;
     'REAL' AE;'CODE';
        .
        .
```

All formal parameters of type _integer_ or _real_ should be specified as _value_.

Procedures: The following describes each procedure in the library and how to use it.

WRITE(dsn,fe,ae)

This procedure transmits the value of the arithmetic expression "ae" to the data set specified by "dsn". The format expression "fe" is of type _integer_ and provides the number of a layout, which itself specifies the particular format required in the output. The form and meaning of the layout and format are explained below.

Examples:

```
     WRITE(1,FORMAT('('-DDD.D')'),X**X)
     WRITE(N,F1,A+B)
```

The layout:

The layout provides a picture for the number that is to be printed. The picture is constructed as follows:

digits	- wherever a digit is required in the output field, a letter D is placed in the corresponding position in the layout. If the letter N is placed in the first digit position, then leading zeros are suppressed. Zeros in the units position and to the right of the decimal point are never suppressed.
sign	- the sign + inserted before all D's and the decimal point will ensure that either + or - appear in the result as appropriate. When N appears in the layout, the sign is floated. The sign - inserted in the layout has the same effect as + except that a space is inserted instead of + for positive numbers. If there is no sign, then no sign is printed.
decimal point	- the decimal point is inserted in the appropriate position when required for the output of real numbers.
space	- a sequence of S's appearing at the start of the layout produces that many leading spaces in the output. For example, the sequence SSS may be replaced by 3S.
exponent	- a floating-point number as input to an ALGOL program or as a constant in ALGOL will be printed if the layout consists of a mantissa and an exponent, separated by an apostrophe. The exponent is an optional sign followed by DD. Any symbol in the layout following the exponent must be a terminator, e.g., D.DD'-DD.

terminators - if required, the layout may be concluded by one of the following symbols; the output is then terminated as indicated:

;	a semicolon is printed in the position specified.
C	a carriage return line feed (newline) is done.
P	a page skip is done after the line is printed.

Nine combinations of terminators are allowed:

; C P CC CCC ;P PC PCC PCCC

The format:

The format expression provides a means of calling a particular layout. Using the KDF9 standard procedure FORMAT, it is possible to associate an integer with a layout. Thus, using LAY to stand for the layout, FORMAT(LAY) will provide the integer corresponding to LAY.

The procedure FORMAT is an _integer_ procedure with a _string_ parameter.

Example:

```
     WRITE(1,FORMAT('('2S+DDD.DD;C')'),A(/I/))
```

Whenever the same layout is to be used for output of more than one number, it is advantageous to assign the integer value produced by the integer procedure FORMAT to an integer variable and use the variable as a format expression.

Example:

```
     F := FORMAT('('NDDC')');
     FOR I := 1 STEP 1 UNTIL N DO
     WRITE(1,F,LIST(/I/))
```

Errors and restrictions in WRITE:

The following restrictions apply to the layout:

(1) The number of leading spaces specified by S's is limited to 15.
(2) The number of digits in the integral

part is limited to 17.
(3) The number of digits in the fractional part is limited to 16.
(4) Blanks may not appear in the layout except in the leading positions or as a terminator.

Error numbers are generated and may be interpreted as follows:

1	Too many spaces
3	Symbol repeated incorrectly
4	Too many digits
5	Wrongly positioned character
7	Unknown character

In addition, the numbers to be printed must be single-precision. An error message is produced if the number is the wrong size for the format specified.

WRITE TEXT(dsn,st)

This procedure prints the text in the string "st" on the data set specified by "dsn". Editing symbols C, P, and S, possibly preceded by an integer repetition factor, and enclosed by additional string quotes, may be inserted in the string to produce the effect of C (newline), P (page skip), and S (space), respectively. The integer before one of these letters specifies the number of such carriage controls.

Alternatively for space, one or more asterisks may be inserted in the text without additional string quotes; blanks in the string are ignored. This means that the asterisk may not occur as a character of the text to be printed.

Example:

```
WRITE TEXT (1,'(''('P')'RESULTS'('C7S')' X*=**')')
```

COPYTEXT(dsnin,dsnout,st)

This procedure copies strings from the first data set specified by "dsnin" to the second data set specified by "dsnout". The parameter "st" is a string consisting of one or two symbols. In the case that this string is a single symbol, copying proceeds from the current symbol on "dsnin" to the first occurrence of this symbol; for two symbols, copying starts immediately after the occurrence of the first symbol and ceases on the occurrence of the second. The symbols of the third parameter are not copied. Blanks are copied by this procedure and asterisks are converted to blanks.

Example:

```
COPYTEXT(0,2,';%')
```

Note: Only symbols that are ALGOL basic hardware symbols may be written or copied by these procedures. The basic hardware symbols are:

```
letters: A-Z
digits:  0-9
symbols: + - * / , . ' = ( ) < > _ | & : ;
```

SPACE(dsn,n)

This procedure increments the character pointer on data set "dsn" by "n" (prints "n" blanks).

GAP(dsn,n)

This procedure is like SPACE except that it starts on a new line; it may straddle a line boundary and it is only available on output devices.

NEWLINE(dsn,n)

This procedure increments the record pointer by "n" ("n" new lines).

OPEN(dsn)

This procedure opens the data set "dsn" and sets up a page size of 60 lines and a line size of 120 characters. In addition, it opens certain workfiles used by WRITE TEXT and COPYTEXT and should be invoked to open the data set before a call is made on either of these procedures.

CLOSE(dsn)

This procedure closes the data set "dsn".

Example: The sample program on the following page illustrates the use of the above procedures and the output produced.

```
$RUN *ALGOL SPUNCH=ALGOLPROG
'BEGIN' 'COMMENT' SAMPLE OF NUMERIC FORMATS;

'COMMENT' DECLARATION OF KDF I/O PROCEDURES;
'PROCEDURE'WRITE(D,F,L);'VALUE'D,F,L;'INTEGER'D,F;'REAL'L;'CODE';
'INTEGER''PROCEDURE'FORMAT(S);'STRING'S;'CODE';
'PROCEDURE'WRITE TEXT(D,S);'VALUE'D;'INTEGER'D;'STRING'S;'CODE';
'PROCEDURE'NEWLINE(D,N);'VALUE'D,N;'INTEGER'D,N;'CODE';
'PROCEDURE'SPACE(D,N);'VALUE'D,N;'INTEGER'D,N;'CODE';
'PROCEDURE'GAP(D,N);'VALUE'D,N;'INTEGER'D,N;'CODE';
'PROCEDURE'OPEN(D);'VALUE'D;'INTEGER'D;'CODE';
'PROCEDURE'CLOSE(D);'VALUE'D;'INTEGER'D;'CODE';
'COMMENT' END OF KDF I/O DECLARATIONS;

'PROCEDURE'DISPLAY(LAYOUT);'STRING'LAYOUT;
'BEGIN'
'COMMENT' OUTPUTS THE THREE NUMBERS WITH SPECIFIED LAYOUT;
'INTEGER'INTF;INTF:=FORMAT(LAYOUT);
NEWLINE(1,2);OUTSTRING(1,LAYOUT);
SYSACT(1,2,22);WRITE(1,INTF,FIRST);
SYSACT(1,2,42);WRITE(1,INTF,SECOND);
SYSACT(1,2,62);WRITE(1,INTF,THIRD)
'END' DISPLAY;

'REAL'FIRST,SECOND,THIRD;
FIRST:=-3.141593;SECOND:=0;THIRD:=1007.999;
OPEN(0);OPEN(1);'COMMENT' OPEN DATA SETS, AT THE SAME TIME WORK
 FILES USED IN WRITE TEXT AND COPY TEXT ARE OPENED;
NEWLINE(1,1);SPACE(1,22);COPY TEXT(P,1,'('<>')');
GAP(1,29);COPY TEXT(0,1,'('*')');
WRITE TEXT(1,'(''('3C29S')'SAMPLE*OF*NUMERIC*FORMATS'('2C')'
               FORMAT*STRING'('8S')'FIRST*NUMBER'('8S')'
               SECOND*NUMBER'('7S')'THIRD*NUMBER'('2C')'
               STANDARD*I/O')');
SYSACT(1,2,22);OUTREAL(1,FIRST); 'COMMENT' AT COLUMN 22;
SYSACT(1,2,42);OUTREAL(1,SECOND); 'COMMENT' AT COLUMN 42;
SYSACT(1,2,62);OUTREAL(1,THIRD); 'COMMENT' AT COLUMN 62;

'COMMENT' OUTPUT NUMBERS WITH VARIOUS LAYOUTS;
DISPLAY('('+DDDDDD.DDD')');
DISPLAY('('-NDDDDD.DDDDDD')');
DISPLAY('('3SDDDD.DDD')');
DISPLAY('('-NDDD.D'DDC')');
DISPLAY('('NDDD;')');
DISPLAY('('D')');
DISPLAY('('-DDD.DDDD'-DD')');
CLOSE(0);CLOSE(1)
'END'
$ENDFILE
$RUN ALGOLPROG+*ALGOLLIB PAR=CODE=*KDFLIB
<*+++++*USING KDFLIB PROCEDURES++++*>HEADING FROM DATA SET 0*
```

```
                         ++++USING KDFLIB PROCEDURES++++
                             HEADING FROM DATA SET 0

                             SAMPLE NUMERIC FORMATS

FORMAT STRING       FIRST NUMBER       SECOND NUMBER      THIRD NUMBER


STANDARD I/O        -3.141592'+00       0                 +1.007999'+03

+DDDDDD.DDD         -000003.142         000000.000        +001007.999

-NDDDDD.DDDDDD           -3.141593           0.000000        1007.999023

3SDDDD.DDD             0003.142           0000.000           1007.999

-NDDD.D'DDC         -3141.6'03
                                            0.0'+00
                                                             1008.0'+00

NDDD;                  3;                 0;              1007;

D                   3                  0
ERROR IN PROCEDURE WRITE
 - NUMBER WRONG SIZE FOR FORMAT SPECIFIED
NUMBER IS   +1.007999'+03

-DDD.DDDD'-DD       -314.1592'-02      000.0000'+00        100.7999'+01

END OF ALGOL PROGRAM EXECUTION
```

*LINPG

Contents: A program that calls the subroutine LINPG.

Purpose: To provide a convenient way to solve linear programming problems.

Use: The program is invoked by the $RUN command.

Program Key: *EXEC

Logical I/O Units Referenced:
SCARDS - data values for the linear programming problem.
SPRINT - the results of the problem.

Description: See the LINPG subroutine description in MTS Volume 3, System Subroutine Description (May 1973), for a description of the subroutine calling sequence.

Values for M, N, M2, and N2 are read from SCARDS using the IOH360 standard format 4I*. The array A is read, row by row, from subsequent input lines using the IOH360 format 42F*. An 42F* format implies that every 43rd element begins a new block and must be put on a new line. For the example problem,

Maximize $X_1 + 2X_2 + 3X_3 - X_4$

subject to $x_1 + 2x_2 + 3x_3 = 15$

$2x_1 + x_2 + 5x_3 = 20$

$x_1 + 2x_2 + x_3 + x_4 = 10$

the associated data could be entered as follows:

```
4 5 3 0
1. 2. 3. 0. 15. 2. 1. 5. 0. 20. 1. 2. 1. 1. 10.
-1. -2. -3. 1. 0.
```

This program interprets the switch returned by LINPG and prints the appropriate comments and results.

Example:

```
$RUN *LINPG SCARDS=DATA
```

In the above example, the data for the linear programming problem is read from the file DATA, and the results are written on *SINK* (the default for SPRINT).

*LISP

Contents: The MTS version of the LISP 1.5 language.

Use: The interpreter is invoked by the $RUN command.

Program Key: *LISP

Logical I/O Units Referenced:

SCARDS	default system input file (LISPIN). Normally contains LISP expressions to be evaluated and data to be read by the READ function.
SPRINT	default system output file (LISPOUT). Normally receives output from the PRINT function and results from top-level expression evaluations.
GUSER	default error input file (ERRIN). Normally contains LISP expressions to be evaluated after an error occurs.
SERCOM	default error output file (ERROUT). Normally receives messages and expression values during error processing.

Parameters: The following keyword parameters may be specified in the PAR field of the $RUN command. The parameters must be separated by a comma.

FCS=n Specifies the number of pages of virtual memory to be allocated prior to initiation of the first garbage collection. If more than "n" pages are needed, they will be allocated; each allocation will be accompanied by the message "FREE SPACE EXPAND". The default is 25.

GC#=n After each garbage collection, the system will ensure that there are at least "n" LISP cells available (8n bytes), by allocating more space if necessary. The default is 4000.

ERR=n Indicates the initial status of interrupt handling:

n=0	program and attention interrupt traps enabled, i.e., interrupts will be processed by LISP.
n=1	program interrupt trap disabled, i.e., interrupt will be processed by MTS.
n=2	attention interrupt trap disabled, i.e., interrupt will be processed by MTS.
n=3	both program and attention interrupt traps disabled.

The default is 0.

OBJ=n	Specifies the number of hash buckets allocated for the literal atom object list. The greater the number of buckets, the faster the resolution of atomic references should be. An odd number is recommended. The default is 69.
INT=m,n	All integer numeric atoms from "m" to "n" will be stored in an internal array, where they can be quickly accessed. The use of this option requires 8(n-m) bytes of storage. "m" must be less than or equal to "n". If only one value is given, the appropriate missing value is zero, e.g., INT=10 is equivalent to INT=0,10, and INT=-20 is equivalent to INT=-20,0. The default is 0,0.

Description: This is a new LISP system designed at the Mental Health Research Institute, University of Michigan. It contains significant extensions of the original language LISP 1.5, described in

> LISP 1.5 Programmer's Manual, J. McCarthy, et al., MIT Press, Cambridge, Massachusetts, 1966,

and is not necessarily compatible with that language. MTS LISP is completely described in MTS Volume 8, LISP and SLIP in MTS. The novice user is also advised to read an introductory publication, such as

> LISP 1.5 Primer, by Clark Weissman, Dickinson Publishing Co., Belmont, California, 1967.

The file *LISPLIB contains checkpointed versions of the following LISP subsystems:

(1) The LISP compiler.
(2) The hypothetical worlds package.
(3) The LISP editor.
(4) The LISP break package.

Additional components may be added periodically. It will not normally be necessary to specify this file explicitly, since it is implicitly referenced by invoking any of the subsystems it contains.

Example:

```
$RUN *LISP SCARDS=PROGRAM PAR=FCS=100,GC#=10000
```

In the above example, the system input file is PROGRAM. 100 pages may be allocated before the first garbage collection, and 10,000 cells will be available after each garbage collection.

*MACGEN

Contents: The macro-library generator program.

Purpose: To generate a directory for a set of macro definitions.

Use: The program is invoked by the $RUN command.

Program Key: *MACGEN

Logical I/O Units Referenced:
SCARDS - the file containing a set of macro definitions.
SPUNCH - the file which will contain the directory.

Description: An MTS-formatted line directory is produced on SPUNCH for the set of macro definitions read through SCARDS. Entries in the line directory must occupy contiguous lines beginning with line number one (1). A directory terminator is inserted after the last macro definition has been read.

Example:

```
$RUN *MACGEN SCARDS=WATMAC(1000) SPUNCH=WATMAC
```

In the above example, a set of macro definitions is read from the file WATMAC, starting at line 1000; the macro library is written to the file WATMAC starting at line 1.

*MADTOPL1

Contents:	A SNOBOL4 program that accepts as input any number of 7090MAD programs and produces as output equivalent PL/I programs.
Use:	The file *SNOBOL4 is run with *MADTOPL1 attached to I/O unit 5 and a size parameter of 50 (60 for programs over 300 statements).
Program Key:	*EXEC

Logical I/O Units Referenced:

4	- program(s) to be translated.
5	- translator (*MADTOPL1).
6	- SNOBOL4 statistics (set to *DUMMY*).
7	- translator output with comments (unless NOLIST option is specified).
8	- PL/I program output without comments (only if PUNCH option is specified).

Operation: This program will translate the following statement types:

assignment statement
implicit function call
statement label

```
     BOOLEAN                  OTHERWISE
     DIMENSION                PRINT BCD RESULTS
     END OF CONDITIONAL       PRINT COMMENT
     END OF FUNCTION          PRINT FORMAT
     END OF PROGRAM           PRINT RESULTS
     ENTRY TO                 PROGRAM COMMON
     ERASABLE                 PUNCH FORMAT
     EXECUTE                  READ AND PRINT DATA
     EXTERNAL FUNCTION        READ BCD TAPE
     FLOATING POINT           READ DATA
     FUNCTION RETURN          READ FORMAT
     INTEGER                  REFERENCES ON (OFF)
     INTERNAL FUNCTION        TRANSFER TO
     LISTING ON (OFF)         VECTOR VALUES
     NORMAL MODE IS           WHENEVER
     OR WHENEVER              WRITE BCD TAPE
     THROUGH - including FOR VALUES OF
```

All other statement types will be flagged. All abbreviations are accepted.

The following restrictions exist:

(1) Format variables cannot always be translated.
(2) No mode numbers will be translated.
(3) Only blocks that have constant subscripts and are in I/O or VECTOR VALUES statements will be translated.
(4) Integers and character strings may not be put in the same array in PL/I.
(5) Format lists may not be read in as data since they need translation.
(6) Only the characters +,-,0,1,2, and 4 will cause carriage control.
(7) The following operators will not be translated: .N.,.LS.,.RS.,.A.,.V.,.EV.,.EXOR.,.THEN.,.EQV., and all programmer-defined operators.
(8) All iterated expressions must be translated by hand.

Options: This translator can be run with several options using a $OPTIONS card at the beginning of each program with the format:

```
     $OPTIONS = option1, option2, ...
```

The options are:

BCD - causes translator to accept BCD-coded decks (MX). (EBCD code is default.)
PUNCH - causes PL/I source deck to be output to I/O unit 8.
NOLIST - stops PL/I program listing and comments output to I/O unit 7.
PRECISE - causes all floating-point variables to be translated into long precision variables in PL/I.
ARRAYS(a1,a2,...) - specifies vector names for which a "(0)" subscript should appear after every nonindexed reference. (ARR and ARR(0) have the same meaning in MAD but not in PL/I). If no parameter list is put after the word ARRAYS, then a "(0)" will be placed after all nonindexed vector names that have been dimensioned in a DIMENSION or any mode declaration statement.

The $OPTIONS card may have any number of continuation cards, each of which must have a "-" sign in column 1. The options will remain in effect until the next occurrence of a $OPTIONS card at which time the default options are reset and then changed according to the new card.

```
     $OPTIONS = BCD, ARRAY(VEC)
```

The above example tells the translator that the MAD programs to follow are BCD coded and causes a "(0)" to be placed after every occurrence of the name VEC that does not already have a subscript.

Comments: Since MAD simple I/O data is incompatible with PL/I simple I/O data, this translator contains a data translator that can be invoked at any time during translation by the occurrence of a $DATA card. All cards following this card are assumed to be data cards up to the next occurrence of a $-card or end-of-file. These cards will be translated to equivalent PL/I simple I/O data cards and will be placed in the file with the PL/I program and listed with the output.

Any number of programs or external functions may be run in succession but must be separated by one or more $-cards (any MAD $-card or the $OPTIONS card will do).

Since the PL/I subroutine library is so different from the MAD subroutine library it will be necessary to rename some of the functions included in the program. For this reason, all of the functions that do not exist or have different names in PL/I are collected with the user-defined functions and put in the function list in the "ENTRY" declaration at the beginning of the translated program. A simple check of this list will show whether or not a function will or will not be prede fined. If a predefined function called by the MAD program is not on this list, then it is also predefined in PL/I.

Note: Since the translation will probably take longer than 30 seconds CPU time, a time estimate should be put on the $SIGNON card for the deck. You should allow about 35 seconds (for the compilation itself) plus 1 second for every statement translated.

Example:

```
$RUN *SNOBOL4 4=MADPROG 5=*MADTOPL1 6=*DUMMY* 7=*SINK*
     8=PL1PROG PAR=SIZE=50
```

In the above example, MAD source programs are read from the file MADPROG, translated, and written to the file PL1PROG (if the PUNCH option is specified).

*NIM

Contents: A program that plays NIM.

Purpose: Demonstration.

Use: The program is invoked by the $RUN command, e.g.,

```
$RUN *NIM
```

Program Key: *NIM

Description: The program is self-describing. It is intended for terminal use.

*PLOT

Contents:	The printer plot program.
Purpose:	To produce plots on line printers and terminals.
Use:	The program is invoked by the $RUN command.
Program Key:	*PLOT

Logical I/O units Referenced:
SCARDS - plot parameters and input data.
SPRINT - output plot.

Parameters: The plot parameters described below may be specified in the PAR field of the $RUN command. The parameters must be separated by commas.

Description: The user is prompted for the number of points to plot followed by the points themselves. Each point is given by its abscissa and ordinate, e.g., the points (1,2.5), (2.1,4.6) would be entered as

```
1. 2.5 2.1 4.6
```

The decimal points are required.

After the plot has been produced, the user is prompted to continue again. By specifying new plot parameters beginning in column 1, a different plot of the same points will be produced. If NEW is specified in the plot parameter, the user will be prompted for a new set of points which will then be plotted. Execution may be terminated by entering an S in column 1.

The plot parameters are:

NHL=n	The number of horizontal lines, n≥2.
NSBH=n	The number of spaces between horizontal lines, n≥1.
NVL=n	The number of vertical lines, n≥2.
NSBV=n	The number of spaces between vertical lines, n≥1.
LABEL='label'	A vertical label centered along the left side of the graph.

NCHAR=n	The number of characters in the label. If omitted, the number of characters in the label string will be used.
LOG=n	n=0 Plot is linear on both axes. n=1 The Y axis is scaled logarithmically. n=2 The X axis is scaled logarithmically. n=3 The X and Y axes are scaled logarithmically.
XSK	The scale factor (powers of 10) for the X axis.
YSK	The scale factor (powers of 10) for the Y axis.
XDP	The number of decimal places in X values.
YDP	The number of decimal places in Y values.
R	The X and Y coordinates are interchanged.
NOR	The R feature is suppressed.
F	Instead of supplying points, the user is asked to specify the name of a file containing a FORTRAN-compatible function. This function will be called to produce points in the interval [XL,XR]. Values for XL and XR should be specified at the same time that F is given. The object records in the file must end with an LDT record, with the function entry point specified.
NOF	The function feature is suppressed.
XL=x	The left boundary value of the plot if F is used.
XR=x	The right boundary value of the plot if F is used.
Y	If F is set, the user must specify values for YH and YL.
YH=y	The highest value of the function that is plotted.
YL=y	The lowest value of the function that is plotted.

NOY	The entire function is plotted. The values used for YH and YL are determined from the values of the function.
CHAR=c	The plotting character is set to "c".
OMIT=n	This will delete of any of the following parts of the graph depending on the value of n. n=1 Delete X grid values. n=2 Delete Y grid values. n=4 Delete the last graph line. Several parts may be deleted by setting "n" equal to the sum of the numbers corresponding to the parts. For example, n=7 would delete all three parts. If one or more functions are to be restored, "n" should be set to the negative sum of the numbers. For example, n=-3 would restore the X and Y grid values.
NEW	New points are to be used in the next plot, or if in function mode, a new file name containing a function to produce the next plot will be requested.

The default cases for the parameters are:

NHL=6,NSBH=1,NVL=6,NSBV=9,LABEL=’ ’,LOG=0,NOF,CHAR=*,NOY

The program does not scale values. Thus, if the user has large (>100,000) or small ($<10^{-3}$) values, the symbol *...* may be printed for axis values. The user can set XSK, YSK, XDP, or YDP to correct for this.

Example: In the following example, input entered by the user is underlined.

```
#$RUN *PLOT
#EXECUTION BEGINS
 HOW MANY POINTS ARE TO BE PLOTTED?
 3
 ENTER ABSCISSA AND ORDINATE OF EACH POINT
 1. 1. 2. 2. 3. 3.



      3.000 +---------+---------+---------+---------+---------*
            I         I         I         I         I         I
      2.599 +---------+---------+---------+---------+---------+
            I         I         I         I         I         I
      2.199 +---------+---------+---------+---------+---------+
            I         I         I    *    I         I         I
      1.799 +---------+---------+---------+---------+---------+
            I         I         I         I         I         I
      1.399 +---------+---------+---------+---------+---------+
            I         I         I         I         I         I
       .999 *---------+---------+---------+---------+---------+

           1.000     1.399     1.799     2.199     2.599     2.999
 IF YOU WISH TO PLOT AGAIN WITH DIFFERENT PARAMETERS,
 PUT THEM IN COLUMN 1
 OTHERWISE PUT AN S IN COLUMN 1
 NHL=2,NVL=2,LABEL='L',NSBV=7



      3.000 +-------*
 L          I   *   I
      1.000 *-------+

           1.000   3.000
 PUT PARS OR S IN COLUMN 1
 S
#EXECUTION TERMINATED
```

*PL360

Contents:	The PL360 compiler.
Use:	PL360 is an ALGOL-like translator that gives the source language programmer much of the power and flexibility of writing assembly code.
Program Key:	*PL360

Logical I/O Units Referenced:

SCARDS - PL360 source statements.
SPRINT - PL360 listings and diagnostics.
SPUNCH - PL360-produced object module.
0 - PL360-produced object module.

Parameters: The following parameters may be specified in the PAR field of the $RUN command. The parameters must be separated by a comma. In each case, the prefix "NO" reverses the definition of the parameter. The default is underlined:

LIST (underlined) / NOLIST	Produce a compilation listing on SPRINT.
LOAD / NOLOAD (underlined)	Produce an object module on logical I/O unit 0.
DECK (underlined) / NODECK	Produce an object module on SPUNCH.

Description: A complete description of PL360 appears in the *Journal of the A. C. M.*, Volume 15, No. 1, January 1968, pp. 37-74.

The file *PL360LIB must be used to supply I/O support for PL360-produced object programs that use the "built-in" standard procedures READ, WRITE, PUNCH, and PAGE. The file is concatenated to the object program on the $RUN command, e.g.,

```
$RUN object+*PL360LIB
```

The following logical I/O unit assignments are used for executing PL360 programs:

SCARDS - referenced by READ
SPRINT - referenced by WRITE and PAGE
SPUNCH - referenced by PUNCH

These logical I/O units must be assigned (or defaulted) if the corresponding procedures are invoked.

Example:

```
$RUN *PL360 SCARDS=PROGRAM SPUNCH=OBJECT
```

In the above example, the PL360 source program is read from the file PROGRAM, and the resulting object module is written to the file OBJECT.

*SCRAMBLE

Contents:	A program for scrambling and unscrambling programs and data in MTS *line* files.
Purpose:	To provide an additional measure of security for user programs and data.
Use:	The program is invoked by the $RUN command.
Program Key:	*SCRAMBLE

Description: To scramble a line file, issue the MTS command:

```
     $RUN *SCRAMBLE
```

Then, in response to the prompting line:

```
     ENTER INPUT FDNAME
```

supply the name of the line file or device you wish to scramble. The program will then prompt:

```
     ENTER OUTPUT FDNAME
```

Supply the file or device to receive the scrambled lines.

Next, *SCRAMBLE will prompt:

```
     ENTER KEYWORD (AND REMEMBER IT!)
```

Supply up to 80 characters of "keyword" to initiate the pseudorandom scrambling process. It is not possible for anyone to decipher the scrambled output with any reasonable amount of effort, so take care not to forget your "keyword" string. *The Computing Center cannot unscramble your output if you forget your "keyword"!*

Finally, the program will prompt:

```
     SCRAMBLE LINE LENGTH TOO?
```

You may supply "Y" or "Yes" to complicate the deciphering still further.

To unscramble a file, run *SCRAMBLE again and supply *exactly* the same "keyword" used to scramble the file earlier. The input FDname will be the file or device containing the scrambled programs or data. The output FDname will be the file or device to receive the unscrambled output. Also, one should respond "N" or "NO" to the prompt for scrambling line length. Otherwise, one will get a random number of "garbage" characters appended to each line.

*SIDEDATE

Contents: The update-comparison program.

Purpose: To compare two *UPDATE command streams which operate on the same base file, producing a merged command stream (optional) and a listing of differences.

Program Key: *SIDEDATE

Logical I/O Units Referenced:

SCARDS	- prompting for input FDnames (conversational mode only).
1	- first update command stream (batch mode only).
2	- second update command stream (batch mode only).
3	- base file (batch mode only).
4	- resulting merged command stream (batch mode only).

Description: *SIDEDATE, as its name implies, does a comparison between two *UPDATE command streams which operate independently on the same base. Both streams are considered to be derived from a common command stream but neither is considered derived from the other. *SIDEDATE optionally produces a merged command stream from the two. (In batch mode, the merged command stream is not optional.) This output command stream will, when given to *UPDATE, include those records which would be included by both command streams, delete those records which would be deleted by either command stream, and insert those records from either command stream which do not conflict with insertions or deletions from the other command stream. An insertion conflict, i.e., an attempt by both command streams to insert different records at the same place, is marked by a comment in the merged output followed by those records from the second command stream. In this case, the second command stream is considered to be "newer" than the first. In the case of an attempt by one command stream to insert records between two records which are deleted by the other command stream, the deletion takes precedence.

The summary of differences between the two command streams may be produced at one of three levels of verbosity: low, medium, or high. (In batch mode, high verbosity is always used.) Low verbosity prints those records which are inserted from one command stream and not the other. Medium verbosity prints, in addition, any command causing the deletion of a record which is included by the other command stream. High verbosity prints, additionally, those records which are deleted by one command stream but not the other.

Limitations: *SIDEDATE cannot recognize similarities in *UPDATE command streams which reposition the base file independently from one another. That is, if a %REWIND or %POSN command appears in one command stream, the same command must appear, eventually, in the other before the command streams can become resynchronized.

Although *SIDEDATE will produce a merged *UPDATE command stream with no apparent inconsistencies, it is up to the user to make sure that the merged command stream is the desired union of the two input streams. This is especially true in the case of an insertion from one stream into a location deleted by the other.

Example:

```
#$RUN *SIDEDATE
#EXECUTION BEGINS

>***SIDEDATE PROGRAM***

>WHERE IS THE FIRST UPDATE DECK?
?deck1
>WHERE IS THE SECOND UPDATE DECK?
?deck2
>WHERE IS THE BASE?
?base
>DO YOU WANT A MERGED UPDATE DECK?
?yes
>WHERE?
?newdeck
>WHERE DOES PRINTED OUTPUT GO?
?*print*
>SET VERBOSITY LEVEL (LOW, MEDIUM, OR HIGH)
?medium

>***DONE***
#EXECUTION TERMINATED
```

The above example illustrates the use of *SIDEDATE in conversational mode. The output from *SIDEDATE is in uppercase; the input from the user is in lowercase.

```
$RUN *SIDEDATE 1=DECK1 2=DECK2 3=BASE 4=NEWDECK
```

The above example illustrates the use of *SIDEDATE in batch mode. The file DECK1 contains the first command stream, the file DECK2 contains the second command stream, the file BASE contains the base file, and the file NEWDECK contains the resulting merged command stream.

*SIMSCRIPT2

Contents: The driver program that invokes the IBM SIMSCRIPT-II compiler contained in the public file *SIM2COMP.

A newer and more reliable version of SIMSCRIPT-II produced by CACI, Inc. is contained in the file UNSP:SIM25. A description of this compiler is available in the file UNSP:SIM25.W.

Purpose: To direct the compilation of SIMSCRIPT-II source programs and perform auxiliary services.

Use: The driver program is invoked by the $RUN command.

Program Key: *EXEC

Logical I/O Units Referenced:

SCARDS	- source statement input to the compiler terminated by an end-of-file condition $ENDFILE (defaults to *SOURCE*@NOCC).
SPRINT	- program listings and diagnostic output from the SIMSCRIPT-II compiler and (optionally) the SIMSCRIPT assembler in *SIM2ASM (defaults to *SINK*).
SERCOM	- error diagnostics for severe errors.
GUSER	- user responses to system prompting messages.
0	- object program output from *SIM2ASM (defaults to the sequential file -LOAD).
6	- the name of the file or device where the compiler is to place the assembler language translation of the source program. If the ASM parameter is used (as is usually the case), then logical unit 6 must be rewindable. Logical unit 6 defaults to the sequential file -SIMU01.

Description: SIMSCRIPT-II "compilation" takes place in two phases: in phase one, *SIM2COMP is invoked to translate SIMSCRIPT-II source statements into 360-assembler-language code; in phase two, *SIM2ASM is invoked to assemble the 360-assembler-language statements into object code.

The file *SIM2LIB contains the subroutine library for execution-time support of SIMSCRIPT-II programs. This file should be concatenated on the $RUN command to the file containing the object modules resulting from the assembly of the 360-assembler language output of the SIMSCRIPT-II compiler, e.g.,

```
$RUN object+*SIM2LIB
```

The file *SIM2ERRORS contains numbered error messages for the errors recognized by the SIMSCRIPT-II library. Each message is located at an MTS line number that is the same as the SIMSCRIPT error number for that message. The message may be obtained via the $COPY command, e.g.,

```
$COPY *SIM2ERRORS(92,92.999)
```

prints a copy of SIMSCRIPT error message 92.

Parameters: The following parameters may be specified in the PAR field of the $RUN command. The parameters must be separated by a comma and/or one or more blanks. The underlined portion of each parameter may be used as an abbreviation for that parameter. Parameters may occur in any order except that if the ASMPAR parameter is used, it must be the last parameter given in the PAR field. If contradictory parameter specifications are given, the rightmost parameter is used.

ASM / NOASM   Default: ASM

The ASM parameter causes automatic invocation of phase two (*SIM2ASM), while NOASM presses phase two. ASM is automatically changed to NOASM if errors occur in phase one.

SOURCE / NOSOURCE   Default: See text

The SOURCE parameter produces a listing of the SIMSCRIPT-II source statements while NOSOURCE suppresses the listing. SOURCE is the default in batch mode; NOSOURCE is the default in conversational mode.

LIST={LEFT|RIGHT}   Default: See text

In printing the source listing, LEFT produces statement numbers on the left side of the listing, while RIGHT produces statement numbers on the right side of the listing. RIGHT is the default in batch mode; LEFT is the default in conversational mode.

ERROR={SHORT|LONG|FULL}   Default: See text

SHORT suppresses the usual printing of long rows of asterisks which accompany error messages, while LONG allows this printing of asterisks. FULL causes the compiler to print a summary of all errors that occurred during compilation. LONG is the default in batch mode; SHORT is the default in conversational mode.

ASMPAR={LIST|NOLIST}   Default: NOLIST

If the ASMPAR keyword is used, it must be the <u>last</u> parameter in the PAR field. All parameters to the right of ASMPAR are passed to *SIM2ASM. The only parameters recognized by *SIM2ASM are LIST and NOLIST.

References:

(1) <u>The Simscript-II Programming Language</u>, P. J. Kiviat, R. Villanueva, and H. M. Markowitz, Prentice-Hall, 1968.
(2) <u>The Simscript-II Programming Language: Reference Manual</u>, Prentice-Hall, 1968.
(3) <u>The Simscript-II Programming Language: IBM 360 Implementation</u>, P. J. Kiviat, H. J. Shukiar, J. B. Urman, and R. Villanueva, Memorandum RM-5777- PR, The RAND Corporation, July 1969.

Examples:

```
$RUN *SIMSCRIPT2
  .
Source program
  .
$ENDFILE
$RUN -LOAD+*SIM2LIB
```

The above example uses all default parameter values and logical I/O unit assignments.

```
$RUN *SIMSCRIPT2 SCARDS=SOURCE SPUNCH=OBJ PAR=ASMPAR=LIST
$RUN OBJ+*SIM2LIB
```

The above example reads the source program from the file SOURCE and writes the object code into the file OBJ. An assembler listing of the intermediate assembler code is produced.

*SKELETON

Contents:	The source for the XPL proto-compiler skeleton.
Program Key:	*EXEC

Description: The skeleton is a minimal framework for the table-driven compiler. The user is expected to replace the syntax tables with appropriate tables produced by the analyzer program (see *ANALYZER in this volume) and complete the compiler by writing the code-synthesizer section. For instructions, restrictions, and details, see

> A Compiler Generator, by McKeeman, Horning, and Wortman, Prentice-Hall, 1970.

*SKELETON is a line file so that sections may be extracted easily. Line numbers are integer-valued starting with 1; hence, the line number in the file and in the compilation listing will coincide.

*SLIP

Contents:	A library of FORTRAN-callable subroutines.
Purpose:	To provide list-processing capabilities for FORTRAN users.
Use:	The file is concatenated on the $RUN command to the object modules in the calling program, e.g.,

```
$RUN object+*SLIP
```

Program Key: *EXEC

Description: The SLIP (Symmetric List Processor) subroutine package is an implementation of Joseph Weizenbaum's IBM 7090 SLIP language. A complete description of the language may be found in the Communications of the ACM, Vol. 6, No. 9, September, 1963, pp. 524-544.

For further information, see MTS Volume 8, LISP and SLIP in MTS.

*STATUS

Contents: The user accounting status program.

Purpose: To print information regarding the user's charge; current and cumulative file space; signons; terminal, plotting, and network time; CPU and wait-memory use; CPU time; I/O; and expiration time.

Use: The program is invoked by the $RUN command.

Program Key: *STATUS

Logical I/O Units Referenced:
SPRINT - listing of the accounting information.

Parameters: The following parameters may be specified in the PAR field of the $RUN command. The parameters may appear in any order and must be separated by blanks and/or commas. The underlined portion of each parameter is the minimum acceptable abbreviation that may be used.

{$|CHARGE|DOLLARS|FUNDS}

Print the remaining amount of funds for the user.

{DISK|FILE}

Print the remaining amount of file space for 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.

PLOTTER

Print the remaining amount of plotter time available to the user.

NETWORK

Print the remaining amount of outbound network connect time available to the user.

EXPIRE

Print the expiration date and time for the user.

FULL

Print all information. In addition to the items listed under NOFULL below, the following quantities are printed: the amount of temporary file space in pages, the cumulative figures for file storage in page-days; CPU and wait-memory in page-hours, CPU time in hours, tape mounts, tape-drive time in hours, lines and pages printed, cards read and punched, paper tape punched and plotter paper used in feet, and batch and terminal sessions; and the expiration date and time. FULL is the default if the program is run in batch mode and no parameters other than HEADING or NOHEADING are specified.

{NOFULL|¬FULL|-FULL}

Print the maximum, used, and remaining figures for the following items: charge in dollars, current file space in pages, concurrent signons, and terminal, plotter, and outbound network time in hours. NOFULL is the default if the program is run in conversational mode and no parameters 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|¬HEADING|-HEADING}

Do not print a heading. If this parameter is specified, it should be first.

Modifiers: One of the following modifiers may be appended to the $, DISK, SIGNONS, TERMINAL, PLOTTER, or NETWORK parameters or their synonyms. If a modifier is to apply to more than one parameter, the parameters may be separated by commas and grouped within parentheses, e.g., ($,DISK)@D.

{@DETAILED|@FULL|@NOREMAINING|@¬REMAINING|@-REMAINING}

Print the maximum, used, and remaining figures for the modified quantities rather than only the remaining amounts.

{@REMAINING|@NODETAILED|@¬DETAILED|@-DETAILED|@NOFULL|@¬FULL|@-FULL}

Print only the remaining amounts for the modified quantities. This is the default if a modifier is not specified.

Description: *STATUS lists on SPRINT information regarding the user's use of computing resources. If the program is run in conversational mode and no parameters are specified (other than HEADING or NOHEADING), the items listed for the parameter NOFULL are printed. If the program is run in batch mode and no parameters 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 in the PAR field of the $RUN command. The information is current at the time the program is run 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 his monthly bill.

Examples:

```
$RUN *STATUS
```

In the above example, the user status information is written on *SINK* (the default for SPRINT).

```
$RUN *STATUS SPRINT=FILE
```

In the above example, the user status information is written to the file FILE.

```
$RUN *STATUS PAR=FULL
```

In the above example, all the status information for the user is written on *SINK*.

```
$RUN *STATUS PAR=$
```

In the above example, the amount of funds remaining in the user's account is written on *SINK*.

```
$RUN *STATUS PAR=NFUL E
```

In the above example, the information that is normally listed when the program is run in conversational mode plus the expiration date and time is printed.

```
$RUN *STATUS PAR=($,D,S)@D,P
```

In the above example, the maximum, used, and remaining funds, file space, and concurrent signons, as well as the remaining plotter time, are printed.

*ST360

Contents:	The student virtual machine simulator.
Purpose:	To simulate a /360 in order to allow elementary software development for small /360 machines.
Use:	The program is invoked by the $RUN command.
Program Key:	*ST360

Logical I/O Units Referenced:

SCARDS	the simulated card reader.
SPRINT	the simulated line printer.
SPUNCH	the simulated card punch.
SERCOM	the simulated operator's console output.
GUSER	the simulated operator's console input.

Description:

The Virtual Machine:

The simulated computer has the following characteristics:

(1) The /360 universal instruction set
(2) No interval timer
(3) No memory protection
(4) 32K bytes of core storage
(5) An I/O complement of:

(a)	an operator's console (1052)	at address 0009
(b)	a card reader (2540)	at address 000C
(c)	a card punch (2540)	at address 000D
(d)	a line printer (1403)	at address 000E
(e)	two tape drives(9-track,2401's)	at addresses 0180,0181

(6) Except at IPL, self-modifying CCW's are not allowed
(7) Instantaneous I/O

These last two characteristics distinguish the virtual machine from a real /360. Programs that run on the virtual machine and that do not make any special assumptions about timing or error occurrence will run on a normal /360. However, any timing dependencies in a program will mean it will not work on both the virtual machine and the real /360.

The Program:

The simulation program can enter a special monitor mode. In this mode, the operator communicates with the monitor rather than with his own program. Monitor mode is entered whenever the virtual machine goes into wait state with no interrupts pending (or interrupts disabled) or whenever the operator generates an attention at his terminal. Monitor mode is indicated by the prefix 'M'. Communication is via GUSER and SERCOM.

When in monitor mode, the following commands can be entered (one-character abbreviations are acceptable):

```
(1)  ATTENTION   - produces an attention interrupt on 0009
                   e.g.,  A

(2)  ATTENTION x - produces  a  device  end  interrupt  on
                   address "x".  "x" must be a legal  hexa-
                   decimal device address
                   e.g.,  A 000C

(3)  CONTINUE    - continue processing
                   e.g.,  C

             GRS
(4)  DISPLAY PSW - displays  (in hexadecimal)  the  desired
             x [y] information.   "x"  must be a valid hexa
                   decimal address (possibly  of  the  form
                   x1+x2+...)   and "y" must be a hexadeci-
                   mal number between 1 and C.   "y"  is  a
                   count  of the number of 4-byte blocks to
                   be displayed.  It defaults to 1.
                   e.g.,  D PSW
                          D 2148+C B

(5)  EXTERNAL    - gives an external interrupt if  external
                   interrupts are enabled.  If they are not
                   enabled, it does nothing (does not stack
                   the interrupt).

(6)  IPL x       - perform  an  initial  program  load from
                   device "x" (which must be a legal  hexa-
                   decimal device address).
                   e.g.,  I 180

             GRx1 y1         modify general register x1 (x1
(7)  MODIFY                - hexadecimal number  between  0
             x2   y2         and F) using hexadecimal  num-
                             ber y1.  The  modification  is
                   done  using a store command.  The second
                   form of the command modifies core  stor-
                   age  starting  at  location  x2 (x2 same
                   format as in DISPLAY command).  In  this
                   form, the modification is done one char-
                   acter  at  a time and commas embedded in
                   y2 are skipped over.
```

```
                     e.g.,  M GR2    25C
                            M 2148+C  0054,108567

(8)  PSWRESTART  - do a PSW restart
                   e.g.,  P

(9)  RETURN      - go to MTS without losing status of
                   virtual machine.  Return to monitor is
                   via a $RESTART command.
                   e.g.,  R

             ON              a command for turning the
(10) TRACE   OFF           - trace feature on and off, and
             LIST [x]        listing the trace table.  Ini-
                             tially the trace feature is
                   off.  If trace is turned on, a two-page
                   (8192 bytes) buffer is acquired.  All
                   interruptions (supervisor calls, program
                   interrupts, I/O interrupts, and external
                   interruptions) and START I/O instruc-
                   tions are recorded.  The format of the
                   trace entries is:

                   a)  SIO entry
                       Flag of 'AA' in byte 0
                       Device address in bytes 2 and 3
                       Contents of CAW in bytes 4-7
                       Contents of instruction counter in
                       bytes 8-11

                   b)  SVC interrupt
                       Old PSW in bytes 0-7
                       New PSW in bytes 8-15
                       Flag of '40' in byte 2

                   c)  Program interrupt
                       Old PSW in bytes 0-7
                       New PSW in bytes 8-15
                       Flag of '80' in byte 2

                   d)  I/O interrupt
                       Old PSW in bytes 0-7
                       New PSW in bytes 8-15
                       Device address in bytes 16-19
                       CSW in bytes 20-27

                   e)  External interrupt
                       Old PSW in bytes 0-7
                       New PSW in bytes 8-15
                       Flag of '20' in byte 2

                   If trace is turned off, the buffer is
                   released (if it exists).  The list com-
```

```
                       mand prints out the trace table.  If "x"
                       (a hexadecimal count of  the  number  of
                       entries  required) is not specified, the
                       whole table is dumped.

                       e.g.,  T
                              ...
                              T LIST C
                              T OFF

                       The first command turns  on  trace,  the
                       second  lists  the  last 12 entries, and
                       the third releases the trace buffer  and
                       turns trace off.

   (11) VARY x text -  mounts  a  tape.  "x" must be either 180
                       or 181 and text must be either a pseudo-
                       device name or a legal  tape  mount  re-
                       quest for a single magnetic tape.

                       V 180 S377 9TP,PNAME=*T*,RING=IN 'TEST'
                       V 180 *T*

                       Multiple  VARY's  on  the  same  address
                       dismount  the  previous  tape.   If  the
                       operator  does  not reply OK, the opera-
                       tor's reply is given on SERCOM.

   (12) ZERO        -  zero all of core
                       e.g.,  Z
```

An end of file in monitor mode on GUSER releases storage and devices and terminates execution.

Error messages are rather terse:

```
 (1)  IPL ERROR    - indicates an I/O error during IPL.
 (2)  SYNTAX ERROR - previous command has a syntax error.
 (3)  DEVICE NOT IN TABLES - from IPL  and ATTENTION commands
                             if the device does not exist.
```

Examples:

```
$RUN *ST360 SCARDS=READER SPRINT=PRINTER SPUNCH=PUNCH
```

In the above example, the file READER is used as the card reader, the file PRINTER is used as the printer, and the file PUNCH is used as the punch. The user's terminal will be the operator's console.

*TABEDIT

Contents: The tab-editing program.

Purpose: To simulate the TAB key on remote terminals that do not support logical tabs and to reformat files based on some character used as a "tab" character. For most common terminals (Teletype, 2741, etc.) a logical tabs feature is supported (see descriptions of the TAB commands in the terminal user's guides in MTS Volume 4, Terminals and Tapes).

Use: The program is invoked by the $RUN command.

Program Key: *TABEDIT

Logical I/O Units Referenced:

SCARDS	the input file to be edited.
SPUNCH	the output file containing the edited lines.
SERCOM	error messages.
GUSER	tab character and tab settings if the PAR field is not given.

Description: The user defines the TAB character and tab settings in the PAR field of the $RUN command. If the first character in the PAR field is nonnumeric, the character will be used as the TAB code, and all occurrences of that character in the input file will be treated as a tabulator key. If the first character in the PAR field is numeric, the normal code for TAB will be used.

The tab stops are entered into the PAR field as column numbers. For instance,

```
     PAR=/,5,10,15
```

specifies "/" as the TAB character, and tab stops at column 5, 10 and 15. An occurrence of "/" in an input line will cause blanks to be inserted up to the next tab stop. If the line pointer is beyond the last tab stop (e.g., in the previous example, column 15 or beyond), one blank will be inserted in the output line for subsequent occurrences of "/".

If a TAB character other than normal TAB is used, occurrences of normal TAB in an input line will not cause tabulation. All occurrences of the TAB character are deleted from output lines.

Examples:

```
$RUN *TABEDIT SPUNCH=EDITFILE PAR=10,16,35,72
```

In the above example, the input is read from *SOURCE* and the output is written to the file EDITFILE. The tabs are set at 10, 16, 35, and 72 (the tab positions often used for a 360-assembly language program). TAB is the tab character.

```
$RUN *TABEDIT SCARDS=AA SPUNCH=BB PAR=%,5,10,15
```

In the above example, the input is read from the file AA and the output is written to the file BB. The tabs are set at 5, 10, and 15 with "%" as the tab character.

*TALLY

Contents: The *TALLY instruction-counting program.

Purpose: To gather execution-time statistics for programs run under MTS.

Use: *TALLY is invoked by the $RUN command. The logical I/O unit assignments and the PAR field should be set up for the program being run under *TALLY.

Program Key: *TALLY

Logical I/O Units Referenced:

GUSER	- input messages to *TALLY.
SPRINT	- statistics collected by *TALLY.
SERCOM	- diagnostic messages.

Description: *TALLY counts the number of times each machine instruction is executed for a program run under its control.

*TALLY prompts the user for the name of the program to be run. The user responds by entering on the first input line the name of the file or device containing the program. *TALLY then asks for a list of options. The user may respond with a blank line if he wants default options or the word FULL. If FULL is specified, *TALLY counts not only occurrences of machine instructions executed by the user's program, but also instructions executed by MTS and system routines such as SCARDS, SPRINT, GDINFO, etc. If no options are specified, only instructions executed by the user's program are counted. In no case does *TALLY count instructions executed by the supervisor.

Programs run under *TALLY may call any of the low-core system subroutines given in the low-core symbol table (LCSYMBOL). If the user's program attempts to branch to an address that is not in this list, the message

```
SIMULATION PREMATURELY TERMINATED
```

is printed along with the address of the branch instruction that attempted to transfer to the routine.

Examples:

```
$RUN *TALLY 5=DATA 6=RESULTS
-LOAD
```

In the above example, the program in the file -LOAD is run under *TALLY; the default options are used. The files DATA and RESULTS are used by the executing program.

```
$RUN *TALLY SPRINT=-LIST
OBJFILE
FULL
```

In the above example, the program in the file OBJFILE is run under *TALLY; the FULL set of options is specified. Output from *TALLY (and perhaps the executing program) is written into the file -LIST.

*TIDY

Contents: A program that renumbers and edits FORTRAN source programs.

Use: The proram is invoked by the $RUN command.

Program Key: *EXEC

Logical I/O Units Referenced:

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

Example:

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

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

Description: TIDY is a FORTRAN program that renumbers and edits other FORTRAN source programs whose statement numbering has become unwieldly and whose readability has deteriorated as a result of the many revisions, patches, and corrections that are typical of reworked programs. TIDY processes programs routine-by-routine and punches new versions of the programs with the following characteristics: (1) all statement num bers increase in consecutive order; (2) only statements referred to by other statements retain statement numbers; (3) all statement number references are updated to conform to the numbering scheme; (4) all FORMAT statements are collected and appear at the end of each routine; (5) all FORMAT and CONTINUE statements that are not referenced are deleted; (6) blanks are interspersed in the FORTRAN statements to improve the readability of the statements, while excessive blanks in the statements are deleted; (7) comments are processed to delete excessive blank comments and to eliminate comments from the FORTRAN statement number and continuation fields; and (8) each card is labeled with a unique letter-number combination. TIDY is entirely written in ASA FORTRAN and accepts and processes all ASA FORTRAN statements as well as some IBM and CDC dialect statements.

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

For the complete description of the TIDY program and the control commands, see CCMemo M10.

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

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

(2) *TIDY deletes "IMPLICIT" statements

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

                         or

               IMPLICIT INTEGER (A-K,Q-Z)
```

would be deleted from the deck.

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

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

would be deleted from the deck.

*TIME

Contents: The object module of the clock program.

Purpose: To print out the current date and the current time.

Program Key: *TIME

Logical I/O Units Referenced:
SPRINT - a single line consisting of the clock time.

Example:

```
#$RUN *TIME
#EXECUTION BEGINS
 CLOCK 15:32:03 DATE 10-16-78
#EXECUTION TERMINATED
```

The above example illustrates the output produced by *TIME. The time is 3:32:03 pm on October 16, 1978.

*UMIST

Contents: The UMIST interpreter.

Purpose: To process UMIST procedures.

Use: UMIST is invoked by the $RUN command.

Program Key: *EXEC

Logical I/O Units Referenced:

SCARDS	input character string.
SPRINT	output after UMIST processing.
SERCOM	error messages and UMIST signon and signoff messages.

Description: UMIST is an interactive text-processing language patterned after TRAC. It interprets strings of characters read one at a time from the input device and prints the value of each string after processing on the output device.

The description of UMIST is given in MTS Volume II (December 1967) and Computing Center Memo 32.

*UPDATE

Contents: The update program.

Purpose: To copy tapes (or files) containing card images, making insertions and deletions.

Use: The program is invoked by the $RUN command.

Program Key: *UPDATE

Logical I/O Units Referenced:

SPRINT	- printed output.
SPUNCH	- output from %PUNCH.
SERCOM	- error messages.
GUSER	- responses to prompting messages.

Commands and insertions are read from the source stream (*SOURCE*). If another source of commands is desired, its FDname should be specified in the PAR field of the $RUN command, i.e.,

```
$RUN *UPDATE PAR=newsource
```

or by using the %SOURCE command. Command and insertion lines must be less than or equal to 80 bytes in length.

Description: *UPDATE is designed to be run in either batch or conversational mode. In batch mode, commands in error, along with any associated insertions, are ignored, and an error total is printed upon termination of the program. In conversational mode, the user is queried when an error is detected. He may enter a new command or continue with the next command.

The update input must consist of 80-column card images which may be blocked to any factor desired. The blocking factor, if greater than 1, must be specified on the %INPUT command. The update output will consist of 80-column card images blocked as specified (except for the last record, and other records which may be truncated by a %CLOSE command). Space for the specified input and output buffering is obtained dynamically when %INPUT and %OUTPUT commands are encountered, and released when %CLOSE is encountered.

All commands take the following form: column 1 must contain a percent-sign "%" which must be immediately followed by the command. Only the first three letters need be given, and they must be uppercase. Most devices are in uppercase mode unless commanded otherwise. Parameters for the command are separated from the command and from each other by one or more blanks (or commas, which are treated _identically_ with blanks). Comments may also be included on a command line after all required parameters. A semicolon ";" is used to indicate the beginning of a comment field, e.g.,

```
     %INPUT FILE1 80 ; THIS DEFINES THE INPUT DEVICE.
```

Lines which are not recognized as commands are treated as insertion lines and are copied immediately to the update output device.

There are four different types of parameters used in the commands: numeric, filemark, character, and keyword. _Numeric parameters_ are used for tape operation counts, deletion counts, etc. They consist of 1 to 20 digits which represent an unsigned decimal integer. A _filemark parameter_ is used to refer to a filemark and consists of the characters FILEMARK or FILEMK. _Character parameters_ are used for FDnames, pseudodevice names, IDs, etc. There are two forms of character parameters. The first form consists of 1 to 80 characters with the restrictions that the first character cannot be a digit or an apostrophe "'". Neither blanks nor commas can be a part of the character parameter. The second form of a character parameter consists of from 1 to 78 characters enclosed in apostrophes, with an apostrophe within the character parameter represented by two adjacent apostrophes. The second form does not restrict the use of a digit or apostrophe as the first character nor the use of blanks and commas within the parameter. Note that the outer apostrophes act only as delimiters and are not considered a part of the character parameter. _Keyword parameters_ are simply keywords which are specified for a specific command, such as ON or OFF.

Examples:

```
     Numeric parameters:     1   2   15   123
     Filemark parameters:    FILEMARK   FILEMK
     Character parameters:   PIL6215   *T*   SEQ.0001
                             '12340001'   'PIL 00001'
                             '''TS"0001'   '*T*(1,100)'
     Keyword parameters:     ON   OFF   *
```

Throughout this description, a device will be described as "open" or "closed". A device is _open_ if it has been allocated a buffer for I/O purposes. A device is _closed_ if its I/O buffer has been released.

Device-Definition Commands

%INPUT incards [n]

"incards" is the pseudodevice name (or FDname) of the file or device to be established for update input. "n" is an integer specifying the blocking factor of the input. If omitted, a blocking factor of 1 card/record (unblocked) is assumed. This command causes "incards" to be opened.

```
Example:   %INPUT *IN* 50
```

%OUTPUT outcards [n]

"outcards" is the pseudodevice name (or FDname) of the file or device to be established for update output. "n" is an integer specifying the blocking factor desired on the output. If omitted, a value of 1 (unblocked) is assumed. This command causes "outcards" to be opened.

```
Examples:  %OUTPUT *OUT* 20
           %OUTPUT FILE1
```

%CLOSE [t]

Device "t" is closed, and if "t" is the output device, the last blocked record (possibly truncated) is written out. If "t" is omitted, the output device is assumed.

%SOURCE FDname

"FDname" is the file or device from which all future commands are read and from which all insertions are made.

Device-Positioning Commands

Since care must be taken not to lose the last (partially) blocked output record, a device-positioning command which is applied to an open device will first implicitly close the device, and then position it.

%REWIND t

Device "t" is rewound. The rewind operation is performed by the calling the subroutine REWIND#. All files or devices which this subroutine can rewind may be specified. See the REWIND# description in MTS Volume 3, _System Subroutine Descriptions_.

Example: %REWIND *OUT*

%FSF t [n]

Tape "t" is spaced forward "n" files. If "n" is omitted, a value of 1 is assumed.

Example: %FSF *IN* 3

%BSF t [n]

Tape "t" is spaced backwards "n" files. If "n" is omitted, a value of 1 is assumed.

Example: %BSF *IN*

%WTM t [n]
%WEF t [n]
%EOF t [n]

Tape "t" has "n" tapemarks (end-of-file marks) written on it. If "n" is omitted, a value of 1 is assumed.

Example: %WTM *OUT*

%FSR t [n]

Tape "t" is spaced forwards "n" records. If "n" is omitted, a value of 1 is assumed.

%BSR t [n]

Tape "t" is spaced backwards "n" records. If "n" is omitted, a value of 1 is assumed.

%LP {ON|OFF}

Label processing is enabled (ON) or disabled (OFF).

%DSN t dsname

The next file on tape "t" is given the data set name "dsname".

%POSN t dsname

Tape "t" is positioned to data set "dsname".

Update Feature Control Commands

%NEWID id [n]

Cards written onto the update output device following this command will have new IDs (columns 73-80). The first card written will have the "id" specified in this command. Succeeding cards will have "id" incremented in steps of "n". If "n" is omitted, the increment defaults to 1. The "id" given in this command should consist of 8 characters. Only the numeric portion of the "id" is incremented.

```
Examples:  %NEWID PIL00001
           %NEWID '00000001' 10
```

%OLDID

Suspends the function described under the %NEWID control command.

%LIST {ON|OFF}

Enables (ON) or disables (OFF) the listing of deleted and inserted cards. This output is written on SPRINT. The default is ON unless SPRINT is assigned to a terminal.

%PUNCH {ON|OFF}

Enables (ON) or disables (OFF) the punching on SPUNCH of all card images sent to the current output device. The default is OFF.

%ECHO {ON|OFF}

Enables (ON) or disables (OFF) the echoing of commands on SPRINT. The default is ON unless SPRINT is assigned to a terminal.

Card Location, Copying and Deletion Commands

Devices to which these commands are applied are required to be open or _implicitly_ closed. A device can be implicitly closed via a device-positioning command or when an end-of-file (EOF) condition has been encountered on that device. If one of the following commands is applied to a device which has not yet been defined via %INPUT or %OUTPUT, or to a device which has been _explicitly_ closed via %CLOSE, the command is in error.

In the execution of these commands, the 360-collating sequence is used for comparisons.

%AFTER id

Copies all cards having IDs less than or equal to "id" from the update input device to the update output device.

```
Examples:  %AFTER PIL03789
           %AFTER '04780000'
```

%AFTER n

Copies the next "n" cards from the update input device to the update output device.

```
Example:   %AFTER 2
```

%AFTER {FILEMARK|FILEMK}

Copies the remainder of the input file to the update output device, leaving the tape positioned immediately _after_ the filemark.

%BEFORE id

Copies all cards having IDs less than "id" from the update input device to the update output device.

```
Example:   %BEFORE PIL07892
```

%BEFORE n

Copies the next "n-1" records from the input device to the update output device.

%BEFORE {FILEMARK|FILEMK}

Copies the remainder of the file and leaves a tape positioned immediately _before_ the filemark. Since a filemark must be read to be sensed, an implicit positioning command is required. Hence, this command should only be applied to tapes.

```
Example:   %BEFORE PIL07892
```

%DELETE id

Copies all cards having IDs less than "id" from the update input device to the update output device, then ignores the card (or cards) having ID of "id" (if any).

Example: `%DELETE PIL00016`

`%DELETE n`

Ignores the next "n" cards on the update input device.

Example: `%DELETE 2`

`%DELETE {FILEMARK|FILEMK}`

Deletes the remainder of the input file, leaving the tape positioned _after_ the filemark.

`%DELETE id1 id2`

Copies all cards having IDs less than "id1" from the update input device to the update output device, then ignores all cards having IDs "id1" through "id2", inclusive, from the input.

Example: `%DELETE PIL00378 PIL00379`

`%DELETE id1 n`

Copies all cards having IDs less than "id1" from the update input device to the update output device, then ignores the next "n" cards on the input tape.

Example: `%DELETE PIL00384 2`

`%DELETE id1 {FILEMARK|FILEMK}`

Copies all cards having IDs less than "id1" to output, and then ignores the remainder of the input file, leaving a tape positioned _after_ the filemark.

`%DELETE * id2`

Ignores all cards on the update input device from the current position up through "id2".

Example: `%DELETE * PIL00778`

`%FIND id`

The update input device is searched for a card with ID equal to "id". The order of the IDs is ignored. Cards passed over are _not_ copied to the update output device.

Example: `%FIND PIL00553`

%FIND {FILEMARK|FILEMK}

Ignores the remainder of the input file, leaving a tape positioned _after_ the filemark.

%UNTIL id

The update input device is searched for a card with ID equal to "id". The order of the IDs is ignored. Cards passed over _are_ copied to the update output device.

%UNTIL {FILEMARK|FILEMK}

Copies the remainder of the input file to output, leaving a tape positioned _before_ the filemark.

Miscellaneous Commands

%COMMENT [comment]

This command is echoed on SPRINT in batch mode for documentation purposes.

%END

This command or an end-of-file encountered in the command stream causes execution of the update program to terminate. All buffers are closed. In batch mode, a count of the errors detected is printed on SPRINT.

%QUIT {ON|OFF}

If enabled (ON) and if errors have been detected during processing, the user is signed off rather than returning to MTS when *UPDATE is terminated. If disabled (OFF), control returns to MTS. The default is OFF.

%IDS

The ID fields of the next input record and last output record are printed on SPRINT.

%MTS

The user is returned to MTS command mode. The program may be restarted with a $RESTART command. Note: This command should not be used for normal program termination since the blocking/deblocking buffers are not flushed and records may be lost.

The %END command should be used for normal program termination.

%MCMD mtscommand

The MTS command specified is executed and control is returned immediately to the UPDATE program.

Sample Command Stream

```
%INPUT    *IN*  50
%OUTPUT   *OUT*   1
%DELETE   PIL00016
%AFTER    PIL00079
            GETSPACE  8193,T=3
            LR    15,1
            L     14,=F'8192'
%DELETE   PIL00830,2
%DELETE   PIL07044,PIL07049
            PUTLINE
            GETLINE
%AFTER    PIL09711
SAVR#       DS   6F
SAVREG#     DS  18F
PARREG#     DS   6F
%AFTER    FILEMARK
%WTM      *OUT*
%REW      *OUT*
%END
```

*XCOM

Contents:	The source for the XPL compiler.
Program Key:	*EXEC
Description:	The XPL compiler is written in XPL. For instructions, restrictions and details, see

> A Compiler Generator, by McKeeman, Horning, and Wortman, Prentice-Hall, 1970.

See also the descriptions of *XPL and *XPLGO in this volume and Computing Center Memo 230, "The XPL User's Guide."

*XCOM is a line file so that sections may be extracted easily. Line numbers are integer-valued starting with 1, hence the line number in the file and in the compilation listing will coincide. It is expected that this file is mainly used to extract procedures to add to the skeleton (see *SKELETON in this volume).

*XPL

Contents: A translator to compile programs written in XPL.

Use: The translator is invoked by the $RUN command.

Program Key: *XPL

Logical I/O Units Referenced:

SCARDS	- source input to translator.
SPRINT	- printed output from translator.
7	- used by the XPL compiler as a scratch unit and also to output the object program. This must be a line file. The default is -XPLFILE1.

The following are usually defaulted, but may be specified:

0	- file containing the XPL object program to be loaded and executed. It must be in standard XPL object program format. Default is *XPLCOMPILER.
1	- used by *XPLCOMPILER as a source library. Default is *XPLIBRARY.
8,9	- used by *XPLCOMPILER as scratch units. Defaults are -XPLFILE2 and -XPLFILE3, respectively. Both of these must be line files.

Parameters: The following parameter may be specified in the PAR field of the $RUN command.

{SIZE|FSA}=n{K|P}	This determines the size of the program's free string area. The suffixes K and P stand for 1024 and 4096 bytes, respectively. If no suffix is given, "n" is the size of the free string area in bytes. The default is 8P.

Description: The file *XPL contains a monitor to load and execute XPL programs. If the program to be loaded is not specified, the XPL compiler is loaded from the file *XPLCOMPILER. *XPL and *XPLGO differ only in where the logical I/O units default to and the amount of working storage acquired. For a description of the XPL language, see

"The XPL Compiler Generator System", by McKeeman, Horning, Nelson, and Wortman, 1968 FJCC Proceedings (AFIPS Vol. 33, Part One), pp. 617-636.

See also the description of *EXPL in this volume and Computing Center Memo 230, "The XPL User's Guide."

Example:

```
$RUN *XPL SCARDS=INFILE 7=OUTFILE
$RUN *XPLGO 0=OBJFILE
```

In the above example, the program in the file INFILE is compiled into the file OBJFILE by the compiler. This program is then executed by the monitor.

*XPLGO

Contents: A monitor to load and execute programs that have been compiled by the XPL compiler.

Use: The monitor is invoked by the $RUN command.

Program Key: *EXEC

Logical I/O Units Referenced:

0	- the file containing the XPL object program to be loaded and executed. This must be a line file. The default is -XPLFILE1.
7,8,9	- scratch units. These must be line files. The defaults are -XPLFILE1, -XPLFILE2, and -XPLFILE3, respectively.

Parameters: The following parameter may be specified in PAR field of the $RUN command.

{SIZE|FSA}=n{K|P}	This determines the size of the program's free string area. The suffixes K and P stand for 1024 and 4096 bytes, respectively. If no suffix is given, "n" is the size of the free string area in bytes. The default is 5P.

Error Codes: The following error codes are issued by *XPL and *XPLGO when unusual conditions are encountered:

200	- End-of-file on program file.
400	- Insufficient core for the program to be loaded. If *XPLGO was being used, try again with *XPL, explicitly specifying unit 0.
500	- Invalid service code from program to monitor.
900	- Invalid output file specified.
1200	- End-of-file error on input files.
1400	- Invalid input file specified.
2200	- End-of-file error on scratch file.

Description: See the *XPL description in this volume for further information.

*1ASR, *8ASR, *9ASR

Contents: These files contain the assemblers for the Digital Equipment Corporation's PDP-1, PDP-5, PDP-7, PDP-8, and PDP-9 machines.

Use: These programs are invoked by the $RUN command.

For PDP-1 assemblies, use *1ASR as the object file.

PDP-5	*8ASR
PDP-7	*9ASR
PDP-8	*8ASR
PDP-9	*9ASR

Program Key: *1ASR for *1ASR
*8ASR for *8ASR
*9ASR for *9ASR

Logical I/O Units Referenced:
SCARDS - input to the assembler.
SPRINT - printed listing from the assembler.
SPUNCH - binary object module.
SERCOM - errror comments.

Parameters: The following parameters may be specified in the PAR field of the $RUN command. The parameters must be separated by commas. They also may be specified during the assembly via the OPTIONS pseudo-op.

LONG - Print long format: 128-character lines, with full 15-bit addresses (PDP-8) and reference listings containing CSIDs.
SHORT - Print short format: 72-character lines, with short 12-bit addresses (PDP-8) and condensed reference listings, omitting CSIDs and MTS line numbers.

ON - Resume printing of assembly listing.
OFF - Suspend printing of assembly listing.

REF - Print normal reference listings; predefined symbols other than #ERROR do not appear.
FULREF - Print full reference listings: all referenced symbols appear.
NOREF - Omit reference listings.

ERR - Write on SERCOM a copy of each assembly listing line that has assembly flags.
NOERR - Inhibit the SERCOM output mentioned above.

NODECK	- Suspend output of object code to SPUNCH.
DECK	- Resume SPUNCH output.
BATCH	- The current assembly may be one of many. After the END card is processed, begin assembling again.

The default parameters are

```
Batch:    LONG,ON,REF,NOERR,DECK
Terminal: LONG,ON,REF,ERR,DECK
```

Description: These files contain the University of Michigan cross-assembler for the PDP-1, PDP-8, and PDP-9 minicomputers manufactured by the Digital Equipment Corporation (DEC). By using these files in conjunction with *8LINK, a user can generate a standard DEC bootstrap tape.

For further details, see Computing Center Memo 329.

Example:

```
$RUN *8ASR SCARDS=PROGRAMS SPUNCH=OBJECT PAR=BATCH
```

In the above example, several PDP-8 assembly programs from the file PROGRAMS are assembled with the resulting object code written to the file OBJECT. The output listings are written on *SINK* (the default for SPRINT).

*1130ASM

Contents: The assembler for IBM 1130 and 1800 source code. This assembler produces object modules and assembly listings with error checking and a full cross-reference table.

Use: The assembler is invoked by the $RUN command.

Program Key: *1130ASM

Logical I/O Units Referenced:

SCARDS - source code and control cards.
SPRINT - assembler listing and error diagnostics.
SPUNCH - object module produced by assembler.
Note: The object modules consist of binary, not EBCDIC card images. If SPUNCH output is written to a file to be punched later, the @BIN modifier must be specified to force column-binary punching when it is punched, i.e.,

```
$COPY OBJECTFILE *PUNCH*@BIN.
```

Parameters: A parameter list may be used, which has the same effect as an *JOB control card. The parameters are the same as the CTL options described below.

Description: Several salient features of the 1130 ASM are:

(1) ISS, ILS, library and user subroutines may be assembled, as well as absolute and relocatable mainline programs. For ISS assemblies, the interrupt level(s) for the ISS must be punched in col(s). 45 (and 50) of the ISS instruction, and the name of the machine (1130 or 1800) must be punched in cols. 21-24.
(2) The 1130 features extended mnemonics are supported.
(3) Literals of two types are permitted in operands.
(4) Address arithmetic in operands includes: addition, subtraction, multiplication, division, and, or. Expressions may be written with up to ten levels of nested parentheses.
(5) The symbol/cross-reference table is keyed by statement number.
(6) A list of numbers of erroneous statements is printed to facilitate locating bad statements in long listings.
(7) Individual and cumulative instruction times may be obtained on the output listing.

Control Cards and Options

Following the $RUN command may come zero or more control cards intended for the assembler. Note that if no control cards are used, a set of default options is assumed, as specified below. Note: Cards anywhere in the input beginning with *, which are not legal control cards, and all cards beginning with / are ignored by the assembler.

The general control card format is as follows:

cc 1	* (asterisk)
cc 2-5	(a keyword that specifies the control card type)
cc 6-12	(other control card data)

The control card types are as follows:

Keyword	Description
JOB	Changes the default options. This card uses the same parameters as the CTL card, and it extends over all programs that follow it. It may be overridden by another *JOB card.
ASM	Specifies the major heading (printed at the top of each page of the output listing) for the assembly. This heading is in addition to and precedes the heading specified on the HDNG statement.
CTL	Specifies assembly options for one assembly. The options are specified as short mnemonics, beginning in cc 6, separated by commas and terminated by at least one blank. The options are listed in the options table below.
SBRK	Causes a sector break in the assembler object module. The binary card currently being built is punched, and a sector break card is punched, according to the standard TSX object format unless the FEAT option is selected, in which case an 1130 sector break is punched. The object module is then resumed. The sector break card type may be specified in cc 6 of this control card. The types are 1, 2, 9, and E, corresponding to the card type placed in the upper half of word 3 of the object card.
EJCT	This causes the output listing to be skipped up to a new page.
TEXT	Page restore is made and the text con-

	tained in cc 6-72 is printed on the printer. When more than one *TEXT cards are present in a row, the restore is done only for the first of the group.
COMMON(nnn)	Specifies the size of common area in the program being assembled. This data is placed in word 5 of a TSX header card. Inside the parentheses, the size of common is specified with an expression, according to the usual rules of address arithmetic.
NAME xxxxx	Specifies the name for the program. This data is placed in words 10-11 of a TSX header card.

SIZE xxx	Specifies the size (in pages) of temporary disk files to be created.
	Default=10 pages
	Maximum=100 pages

DBCT xxx	Specifies the disk block count to be used in object decks produced. This control card overrides the normal calculation of the disk block count performed by the program.

CTL card options:

Option Keyword	Description
NL	No listing (listing is suppressed)
NO	No object (output module is suppressed)
NX	No cross-reference table (Xref/symbol table is suppressed)
NOER(n)	No object module if error count exceeds the specified amount. The amount is specified by a single hexadecimal digit in parentheses after the keyword, according to the following codes:

Digit	Count Implied
0	0
1	16
2	32
3	64
etc.	

OGL	Omit listing-generated code from literals and macros. The assembled data is still placed in the object module.
DS	Double-space the listing.
IT(n)	List instruction times. The times of the instructions are printed in the

right-hand margin of the listing. Single times and cumulative times are given. The cumulative time is increased by the instruction time multiplied by the time factor as specified by the most recent TIME pseudo-macro assembly language statement. If no TIME statement has appeared, the time factor is assumed to be (1.0). The single digit in the parentheses specifies the machine type, according to the following codes:

Code	Machine Implied
0	1130, 3.6 usec memory
1	1130, 2.2 usec memory
2	1800, 4.0 usec memory
3	1800, 2.0 usec memory

LO	Punch loader overlay cards. This option implies that the object module is to be in 1130/1800 key set format.
CI	Core-image object module format desired; implies 1130/1800 key set object format.
CS	Check sequence numbers. The sequence numbers, cc 73-80, are checked to see that each number is greater than or equal to the previous one.
TSX	TSX object format desired.
FEAT	1130 features system object module format desired. Note that this differs from the (assumed) key set object type only in that subroutine calls may be split between two data cards in the features format.
029	The 029 keypunch character set is used exclusively in the source deck; no input translation is necessary.
026	The 026 keypunch character set is used; the following translations are made in the source deck before being read by the assembler:

char	026 punch	trans. to	029 punch
+ or &	12	+	12-6-8
(or %	0-4-8	(	12-5-8
) or ¤	12-4-8	)	11-5-8

Note that this option allows the user to intermix source cards punched on both keypunches, provided he did not intend the punching of the characters (, %, or & in any 029-punched statement.

Assumed Options:

When the *CTL card is absent, or is not otherwise specified, the following options are assumed:

```
     NOER(F),029,FEAT
```

This means that the assembler will produce a single-spaced listing with Xref/symbol table, an object module in 1130 card system format, and will only accept 029 punches for +,), and (.

Example:

```
$RUN *1130ASM SPUNCH=1130OBJ PAR=NX,026,NOER(1)

     (source deck)

$ENDFILE
```

In the above example, an 1130 source program is assembled into the file 1130OBJ; no cross-reference table is printed, no object module is written if the error count exceeds 16, and the source program is assumed to be punched on an 026 keypunch.