REFERENCE MANUAL

DISK BASED OPERATING SYSTEM

GENERAL AUTOMATION, INC.

REFERENCE MANUAL

GA 18/30
DISK BASED
OPERATING SYSTEM

Technical Notice No. 1 Installed

GENERAL AUTOMATION, INC.
Automation Products Division
1055 East Street, Anaheim, California 92805 (714) 778-4800

GENERAL AUTOMATION, INC.

88A00142A-B

REVISION

Symbol	Description	Approved	Date
A	Programming Release	RAD	May 70
B	Revision	RAD	Aug 70

CONTENTS

CONTENTS (continued)

CONTENTS (continued)

SECTION 1
INTRODUCTION

1.1 GENERAL

The GA 18/30 Disk Based Operating System (DBOS) is a comprehensive, user-oriented operating system which provides the User with the following features:

a. System Operation.

Provides efficient operations under monitor control.

Simplifies manual operations.

Reduces operator errors and job set-up time.

Provides simplified control sequences.

Allows efficient file control.

b. Job Processing.

Initiates assemblies, compilation, program check out and execution.

Assigns files and peripheral equipment.

Allocates memory.

Provides batch-processing of jobs.

c. Input/Output System

I/O drivers.

I/O interrupt control.

Data packing and unpacking.

Device independent log ical I/O units and devices.

d. Standard Processor Master Files.

Symbolic assembler.

FORTRAN

Core image converter (loading and linking program).

Source language editor.

Debug routine.

e. Programmer/Operator Aids

Program execution by program name.

Diagnostic error messages.

Simplified calling sequences.

Memory dump.

Directoried files.

f. System Preparation and Maintenance.

Simplified system generation.

Replacement and deletion of directoried files.

Listed output of file directories.

User program in directoried files.

DBOS allows job processing to proceed under the direction of control commands. Control commands may be submitted by the programmer or prepared by the operator and input to the system from the teletype keyboard, card reader, or paper tape reader. Jobs may be batched or singly processed under guidance by the operator.

DBOS operates in the following minimum hardware configuration:

a. A GA 18/30 Industrial Supervisory System Computer with 8192 words of core memory.

b. 1 Model 1362 or 1363 teletype unit.

c. 1 Model 1341 or 1344 Disk storage unit.

The addition of the following peripherals enhances the utility of DBOS:

a. Model 1311 card reader.

b. Model 1313 card punch.

c. Model 1352 line printer.

d. Model 1321 paper tape reader.

e. Model 1322 paper tape punch.

DBOS generation requires high speed paper tape or card input. This manual is intended as a general reference manual to be used by both programmers and operators. It provides descriptions of DBOS processing functions, including control command configurations, and standard processor usage; system operations, including bootstrapping, system entry methods, and system messages; system generation methods and techniques; and usage of the input/output system.

SECTION 2
DBOS PROCESSING

2.1 GENERAL INFORMATION

The GA 18/30 Disk Based Operating System (DBOS) provides complete processing capabilities in the following areas:

a. Assembly and compilation of source language programs.

b. Loading and execution of user programs.

c. Maintenance of user and system programs on disk storage.

d. Device independent input/output operations.

e. Sequential job processing from control commands.

DBOS consists of the following components:

1. The Monitor.

   The monitor is a core resident program which processes internal interrupts, loads system processors from disk, processes user-programmed returns to DBOS, and contains system-wide parameters.

2. The Logical I/O System.

   The logical I/O system is a core resident set of input/output drivers, I/O device tables, and a central control routine for performing operations according to logical device specifications.

3. The Executive.

   The executive is a system processor which is loaded by the monitor to process control commands. The control commands define logical unit assignments, program assembly or compilation, program execution, and disk utility functions.

DBOS is initially created by a system generation process. This process is described in section 4 of this manual. During system generation, the system is written onto disk and , when it is to be activated, is read into core via a disk IPL operation (see section 3, DBOS Operation).

2.2 CONTROL COMMAND INPUT

The DBOS user communicates processing requests to the system by means of control commands. Control commands are read by the system and input to an 80-character storage buffer for processing.

These commands have the following syntactical format:

$command [, option] Δcomments

---

where:

$	is the control command identification character and is always the first character of the control command input record.
command	represents one of the legal command syntax structures.
[option]	represents an optional command modifier. All data within braces is optional including delimiters.
Δ	indicates a blank character. This blank acts as the command terminator. Only comments may follow.
comments	represents an optional string of characters from the GA 18/30 character set and are included for annotation purposes. Comments are listed on the system log but have no processing function.

Control commands are input from the system's logical unit CC. The teletype keyboard is the standard device assigned to logical unit CC. The most meaningful alternate CC device is the card reader since this provides batch-processing capability to the system.

2.2.1 CONTROL COMMAND INPUT FROM THE TELETYPE KEYBOARD

When logical unit CC has its standard device assignment (teletype keyboard), DBOS indicates the start of a new control command sequence with the message

DBOS CC

This signifies that subsequent control commands are to be keyed-in by the operator. The system processes these commands as follows:

1. The system signals its readiness to receive a control command by output of a control command request which consists of a Line Feed, a Return, a ? and a space. Only following this request can the operator key-in a control command.
2. The operator may key-in a control command of up to 80 characters, including comments. Keyed-in control commands are terminated by striking the RETURN key on the keyboard (no trailing blank is required). Processing begins immediately following the RETURN key.

   If comments are included, a blank (space) must appear before the comment string.

3. Key-in errors may be deleted, if detected prior to the RETURN key, in either of two ways:

   a) By striking the RUB-OUT key on the keyboard, the operator deletes the entire command. Upon receipt of the RUB-OUT key, the system immediately requests a control command as in 1., above.

   b) To delete the last n characters of the control command, the operator strikes the left arrow (←) key n times. Each time the ← is struck the last remaining input character is deleted from the input record.

2.2.2 CONTROL COMMANDS INPUT FROM CARDS

When logical unit CC is reassigned to the card reader, the system reads and processes 80-column card image records. The $ character must be in column 1 and all characters up to the first blank constitute the control command.

Once the card reader is assigned as the CC device, it continues as the control command input medium until CC is reassigned or the system is reinitialized.

2.3 CONTROL COMMAND LISTING

Control Commands are always listed, during processing, on the system log (logical unit SL). The line printer is the standard SL device. If SL is assigned to the teletype printer, the listing is suppressed if the CC input is from the teletype keyboard.

2.4 SYSTEM CONTROL COMMAND AND COMMENTARY

The following subparagraphs define the specific control commands acceptable to DBOS. Each must conform to the general format specified in paragraph 2.2. Optional elements are indicated by brackets ([ ]). An optional element includes all items within the braces, i.e., [, P] indicates that the comma and P may be omitted. These commands define the major processing functions of the system.

Commentary

The comment command allows additional commentary in the control command input stream. It is of the form:

$CΔ commentary

Example

$JOB	Assemble real time executive
$C	10/15/70 version 1, modification 2
$A	Invoke assembler

2.4.1 LOGICAL UNIT ASSIGNMENT

Each DBOS system is generated with standard logical unit assignments to conform to the particular hardware configuration. This includes User disk sector allocation. Table 2-1 defines the generated standard assignments. Logical units may be referred to symbolically or by number (0-15).

A logical unit assignment command is provided which can override the generated standard assignments. This override can be defined to continue for the duration of only one JOB or through multiple JOBS. A JOB duration is defined as the period between the occurrence of a $JOB command and a subsequent $JOB command. Programs which use the Logical I/O system for input/output may, through use of this command, use any devices. This command assigns one of the logical units shown in table 2-1 to one of the files shown in table 2-2. It has the following format:

```
$lun=file ┌(name) ┐ [,P]
          └(bs-es)┘
```

where:

lun — represents one of the logical unit names or valid decimal numbers shown in table 2-1.

file — is the name of the device (CR, PR) or "file" (DS, WC) as shown in table 2-2, to which the logical unit is being assigned.

(name) represents a program name if "file" is a directoried file (DS, LB, DC or UL). If (name) is used, the logical unit assignment is to that program within the specified file.

(bs-es) represents a disk storage area. "bs" defines the beginning sector address (in hexadecimal) and "es" defines the ending sector address (in hexadecimal). If this option is used, the logical unit assignment is to that area of disk.

P specifies that a LUN assignment is to be maintained through successive JOBS. Logical units defined with the P option will be overridden by an IPL operation. When the P option is omitted the LUN assignment will be maintained for one JOB sequence only. The next $JOB will reset the LUN assignments to the last user specified LUNs tagged with the "P" option. LUNs not specifically specified by the user will be reset to the generated system standard.

Examples:

1. $LO=TY

   The teletype printer is assigned as the listing output device.

2. $CC=CR

   The card reader is assigned as the control command device.

3. $SI=WS

   The working source language data file (WS) is assigned to symbolic input; i.e., subsequent symbolic input will be taken from the WS file on disk.

4. $SI=DS (USRI)

   The symbolic input logical unit is assigned as program USRI which is in the directoried source file (DS).

   Note: Caution must be exercised when assigning non-standard devices to OM, CC and SL. For example; assignment of the line printer to OM to list FORTRAN error messages is inadvisable. FORTRAN expects to receive input from device OM which is not possible from a line printer. Caution should always be exercised not to assign output only devices as input files and vice versa.

2.4.2 JOB COMMAND

The JOB command sets the logical unit assignments to the generated standards (table 2.1) or to the last user specified assignments tagged with the "P" option. (see 2.4.1) In addition the JOB command initializes (opens) disk files. This command may be used at any time, but it normally is the first command in a job stack.

The format is:

```
$JOB
```

2.4.3 COPY COMMAND

The COPY command provides file-to-file copying capability. It also provides a method for defining names of programs in directoried files (DS, LB, UL and DC files).

The format of the COPY command is:

$$\text{\$COPY}\left[\ ,\ \begin{matrix}A\\B\\C\\D\end{matrix}\right]\underbrace{,\ \text{file}_1\begin{bmatrix}\text{(name)}\\\text{(bs-es)}\end{bmatrix}}_{\text{Source}}\left[\ ,\ \begin{matrix}A\\B\\C\\D\end{matrix}\right]\underbrace{,\ \text{file}_2\begin{bmatrix}\text{(name)}\\\text{(bs-es)}\end{bmatrix}}_{\text{Destination}}$$

Where:

$\begin{bmatrix}, & A\\, & B\\, & C\\, & D\end{bmatrix}$ are optional data type specifications

A- specifies ASCII data in 80 character record form.

B- specifies binary data in 54 word assembler object or core image record form.

C- specifies unchecksummed binary data in 60 word record form

D- specifies binary data in 320 word record form on disk.

, $file_1$ represents the input (source) file involved in the COPY

, $file_2$ represents the output (destination) file involved in the COPY

[(name) (bs-es)] represent options, and (name) and (bs-es) have the same meaning as defined in subparagraph 2.4.1.

When the COPY command is processed, data is copied from $file_1$ to $file_2$ until an ASCII END, binary end (OFOO) record, end-of-data ($EOD), or end-of-file is encountered. An ASCII END may exist as the first or second field of a source record. Therefore a label may appear before the END record in an' assembler source $file_1$, i.e., NAME END. NOTE: The END statement may not begin in columns 1 or 21.

If A, B, C or D is not specified, the data type is determined from the file name. Disk files are assumed to be 320 word binary records. If the file is capable of maintaining both ASCII and binary data, ASCII is taken as the normal type.

The DBOS system maintains implicit definitions for all files, whether disk resident or external devices. The optional type specification overrides these implicit definitions, i.e., $COPY,CR,A In this example the implicit definition of type binary for file LB is overriden and forced to be ASCII. Such action, while permissible, will create improper structure of the file for the inserted program (LB may contain only type "B" data). The user should only specify a type code when:

1. The destination file is of a "type" different from the input file "type".
2. An external file of unchecksummed binary data is to be copied, type "C".
3. Both files are capable of containing ASCII and binary data and type "binary" is required. Note: the default type for such files is ASCII. (i.e., CR, CP)

LIBRARY EXPANSION

When a program is copied to a directoried library file (LB or UL), the program name(s) is taken from the input program. Multiple programs may be input to create or append a library. The input must be terminated with an end of data ($EOD) record. (See section 2.4.12 for use of $EOD.)

WC AND DC INPUT

Core image data are absolute programs with subroutines properly linked and all external references satisfied. In order for programs to be in core image format, they must have been processed by the core image converter (see subparagraph 2.5.3) which loads binary object programs, links program elements and performs proper relocation adjustments, or the assembled binary object from an absolute assembly. No external references may be used in assembly. Precede $A with $BO=WC to create object in WC file.

Once a program is in core image format, it can be loaded from the DC or WC file and executed without further processing. (See paragraphs 2.4.20 and 2.4.11).

$COPY Examples:

1. $COPY, DS (PROGA), WS

   Copy PROGA from the DS file to the WS file. ASCII 80 character record form is used.

2. $COPY, B, CR, LB
   $EOD
   Copy a binary file from the card reader to the LB file. The binary file is identified in the LB directory according to the name contained on the input program data records. (B specification is redundant but permissible.)

   The last copy operation affecting a library must be followed by a $EOD.

3. $COPY,WC,DC(PROGX)

   Copy the working core image file (WC) into the directoried core image program data file and give the program the name PROGX in the DC directory.

4. $COPY,DS(PROG5),LP

   Copy program PROG5 from the DS file to the line printer (LP).

5. $COPY,DC(NAME),B,TY

   COPY program name from DC to the teletype.

2.4.4 REPLACE COMMAND

The REPLACE command provides the capability to replace a previously named program in a directoried file with data from another file.

The format of the REPLACE command is:

$$\text{\$REPLACE,file}_1\underbrace{\begin{matrix}(\text{name})\\(\text{bs-es})\end{matrix}}_{\text{destination}}\begin{bmatrix}A\\,B\\C\\D\end{bmatrix}\underbrace{,\ \text{file}_2\begin{bmatrix}(\text{name})\\(\text{bs-es})\end{bmatrix}}_{\text{source}}$$

where:

,file_1	represents the output file, i.e., the file which will receive the replacement program.
(name)	represents the name of the program in file_1 which is being replaced. The name will be assigned to the new program unless file, is a library file. In this case the program name will be taken from the input data.
$\begin{bmatrix},A\\,B\\,C\\,D\end{bmatrix}$	are optional data type specifications, as defined in subparagraph 2.4.3.
,file_2	represents the input file; i.e., the file from which the replacement program is to be copied.

(name) are optional and have the same meanings as defined in

(bs-es) subparagraph 2.4.1.

$REPLACE examples:

1. $REPLACE,DC(PROG7),WC

   Replace PROG7 in the DC file with the program in the WC file.

2. $REPLACE,LB(SUBA),CR
   $EOD
   $PACK
   Program SUBA of the LB file is replaced by the binary record in the CR file. The last REPLACE operation affecting a library file must be followed by a $EOD and $PACK.

3. $REPLACE,DS(PROG23),A,WS

   Program PROG23 of the DS file is replaced by the symbolic program in the WS file.

NOTE

The REPLACE command does not write the new program over the program being replaced. Rather, the new program is written in the first unused space of the file and the "bs" and "es" addresses in the directory are altered to reflect the replacement. A subsequent $LDIR command would indicate the area occupied by the replaced program as ****** unless a $PACK command has compressed the file. See $LDIR (2.4.6) and $PACK (2.4.7). After replacing a library routine a $PACK must be executed. (See 2.4.7.)

2.4.5 DELETE COMMAND

The DELETE command is used to delete an existing named program from a directoried file.

The format of the DELETE command is:

```
$DELETE, file (name)
```

where:

, file	represents the name of one of the directoried files (DS, LB, UL or DC)
(name)	represents the name of the program which is to be deleted.

Example:

```
$DELETE, LB (SUBA)
$PACK
```

The program named SUBA is deleted from the LB file. Note: A $PACK command must be executed after deleting subroutines from the library. See 2.4.7.

2.4.6 LIST DIRECTORY COMMAND

The list directory command causes the current status of a directoried file to be listed on the SL (system log) unit.

The command format is:

$LDIR, file [bs-es]

---

where "file" is the name of the directoried file whose directory is to be listed.

The LDIR listing has the format:

bs es type rpr ipr name

for each item in the directory. "bs" specifies the beginning sector address (in hexadecima l), and "es" the ending sector address (in hexadecimal), of the disk area assigned to the storage of program "name". For library subroutines type indicates the program type as

LIB	Called by LIBF
ENT	Called by CALL

"rpr" indicates the precision of real numbers and "ipr" indicates the precision of integer numbers as

blank	Unspecified
SPR	Standard precision
EPR	Extended precision

If an area has been deleted or replaced, the listing format is:

```
bs es ******
```

2.4.7 PACK DIRECTORIED FILE COMMAND

The pack directoried file command packs the elements in directoried files to eliminate unused sectors resulting from program deletions or replacements.

The format of the command is:

```
$PACK, file
```

where "file" is the name of one of the directoried files (DS, LB, UL or DC).

NOTE

For proper processing of libraries (LB or UL) by the Core Image Converter . The PACK command must be issued following any update to the library.

2.4.8 DUMP COMMAND

The dump command produces a hexadecimal listing of a file on the SL (system log) unit.

The command format is:

```
          ┌ A ┐          ┌(name) ┐
          │ B │          │       │
$DUMP     │,  │  , file  │       │
          │ C │          │       │
          └ D ┘          └(bs-es)┘
```

where:

[,A ,B ,C ,D]	are optional data type specifications as defined in subparagraph 2.4.3, and define the data type of the file data.
,file	is the name of the file to be dumped
[(name) (bs-es)]	are optional and have the same meanings as defined in subparagraph 2.4.2.

A $DUMP listed line has the format:

loc data$_1$ data$_2$ ... data$_n$

where:

loc represents the relative record location (hexadecimal) of the first data item in the first data item in the line.

$data_i$ represents a data item

n is 16 if SL is assigned to the line printer and 8 if the SL is assigned to the teletype printer.

If the nth item of a line (8th or 16th item) and the next n items are identical, the next line is not printed. When this occurs one response occurs.

Assume a block of 32 words contains a recurring series of numbers; 1, 2, 3, 4, 5, 6, 7, 8 followed by a disimilar pattern 8, 7, 6, 5, 4, 3, 2, 1.

The listed output would appear as two lines as follows:

LOC	$Data_1$	$Data_2$						$Data_8$
1	1	2	3	4	5	6	7	8
33	8	7	6	5	4	3	2	1

examples:

```
$DUMP,DC(PROGA)
```

Dumps the program named PROGA from the DC file.

```
$DUMP,B,DK(15F0-15F5)
```

Dumps the binary data from the DK file, sectors 15F0 through 15F5.

2.4.9 PDUMP COMMAND

When the monitor assumes control of the system, core memory is saved prior to execution of the system executive. The PDUMP command provides a selective dump of this core memory in hexadecimal.

The command is:

$$\$PDUMP \left[, loc_1 - loc_2 \right]$$

where:

$[, loc_1$	represents the location (in hexadecimal) of the first core memory word to be dumped.
$-loc_2]$	represents the location (in hexadecimal) of the last core memory word to be dumped.

Output resulting from PDUMP is on the SL (system log) unit and consists of lines in the following format:

$$loc \quad data_1 \quad data_2 \ldots data_n$$

where:

loc — the memory location (in hexadecimal) of the first data item in the line. "loc" is always a modulo 8 address.

$data_i$ — represent the contents of mem ory location "loc_2" through "loc_n".

n — is 16 if SL is assigned to the line printer and 8 if SL is assigned to the teletype printer.

If the nth item of a line (8th or 16th item) and the next n items are identical, the next line is not printed. When this occurs, one upspace occurs. See section 2.4.8 for listing example.

If the optional [, loc_1 - loc_2] is not present in the PDUMP command, the entire core memory is dumped.

The system saves the program registers in the following locations:

REGISTER	LOCATION (HEX)
I	/77
A	/78
Q	/79
Index 1	/7A
Index 2	/7B
Index 3	/7C

examples:

1. $PDUMP,15A5-15B3
   Dumps the contents of saved core locations /15A0 through /15BF.

2. $PDUMP,70-7F

   Dump the contents of saved core locations /70 through /7F which includes the program registers.

NOTE

The monitor performs a core save function everytime it is entered by a functional program. Control may be returned to the monitor in any of several ways:

1. Manual entry at locations /71 or /73.
2. Programmed return via a
   a. CALL MON normal return
   b. CALL MONE error (abort) return
   c. FORTRAN CALL EXIT or STOP.
   d. Console interrupt.

Refer to sections 3.1.2 through 3.1.6.

2.4.10 PROGRAM EXECUTION COMMAND

A program which is contained in core image format in the working core image file (WC) may be loaded and executed by use of the following command:

$LOAD [, $name_2$]

---

where "$name_2$" is optional and represents the name of a second program from the DC file which is loaded along with the program from the WC file. If "$name_2$" is present in the command, program "$name_2$" is executed when loading is complete. If "$name_2$" is not present, the program from the WC file is executed.

Only one program may be contained in the WC file at a time.

This command is similar to the Processor execution command (2.4.11) except that it allows loading unnamed programs.

A program may be placed in the WC file by a $COPY operation.

Example:

$COPY,WB,WC would copy the object output from the assembler in file WB to the core image file WC. $COPY, CR, WC would copy an object deck to WC for execution.

The following command sequence will result in an executable program in file WC from source media.

```
$JOB
$A and $F
Source Statements
$EOD
$CIC
[*MAP]
*BUILD
$LOAD     Immediate execution or
$JOB
$BO=WC    assembler object output to WC
$A
$LOAD     immediate execution
```

A second program may be called into core from the DC file along with the program from WC.

The format:

```
$LOAD,PROG2
```

where PROG2 is a program stored in the directoried file DC.

The second program must be origined such that it does not overlay the program loaded from WC. (See BOUND directive under CIC, section 2.5.3.) (An assembler ABS and ORG directive may be used to origin an assembly language program.)

A typical use for the double program call is in debugging. The command

```
$LOAD,D
```

will load the program from WC and the DBOS debug routine from DC. Execution will begin with debug (see section 2.5.5).

Note: The debug program occupies /700 locations of high core. (origin /7900)

2.4.11 PROCESSOR EXECUTION

Any program may be specified as a processor by having that program stored in the directoried core image file (DC) with its name in the DC directory. A number of predefined processors are included in DBOS. These are defined in paragraph 2.5.

A user may add his programs (processor) to the DC file by use of the copy command.

Examples:

$COPY,CR,DC(NAMEX)	Copy an external program into DC from cards.
$COPY,WC,DC(PROG1)	Copy the unnamed program in WC into the DC file and call it PROG1.

Programs which are in the DC file may be loaded and executed by use of the following control command:

$\$\text{name}_1 \left[,\text{name}_2\right]$

where:

$name_1$	represents the name of the program (processor) to be loaded.
[$,name_2$]	is optional and represents the name of a second program from the DC file which is also to be loaded.

If "$name_2$" is specified both programs are loaded into core memory and program "$name_2$" is executed.

Normally, "$name_2$" is a debug program.

Example:

1. $PRG15

   Load and execute program PRG15.

2. $PRG15,D

   Load both program PRG15 and D (the debug program) and execute the debug program. See section 2.4.10 for examples and origin restrictions.

2.4.12 WRITE END-OF-DATA IMAGE ($EOD)

The Core Image Converter, CIC, (see section 2.5.3) will accept data from up to four separate binary files. Each file must terminate with a $EOD image record. This image causes the CIC to terminate access of one file and advance to the next or if the file is LB to stop accessing completely.

The command $EOD will close a file.

Example:

$JOB	
$F	Fortran mainline to file WB.
$A	Assembler output to file WB.
$EOD	Close WB file. No more data may be entered into WB.
	NOTE: The $EOD command writes on logical unit BO. The standard assignment for BO is WB.

When a library is being terminated the $EOD command writes into the library file instead of BO.

Example:

```
$JOB              open files
$COPY,CR,UL(bs-es) build library starting at bs.
$EOD              terminate UL
$JOB              reset limits for UL to standard
```

To make use of this new library the following steps might be used.

```
$JOB              open files
$F                compile mainline
Source Statements
$EOD              close WB file
$SB=DP(bs-es)     set disk limits for new SB
$CIC              call core image converter
*BUILD,SB         build program and include ALL data in SB in program

$JOB              open files
$F                compile mainline
Source Statements
$EOD              close WB file
$UL=DP(bs-es)     set disk limits for new UL
$CIC              call convert
*BUILD,UL         build program using only those routines called by mainline
                  from UL.
```

$JOB	open files
$F	compile mainline
Source Statements	
$A	assemble subprogram
$EOD	close WB file
$SB=DP(bs-es)	set disk limits for SB
$CIC	call convert
*BUILD,SB,UL	build program using all data from new SB and required routines from UL.

NOTE: In all cases the standard library file, LB is used to complete build process.

2.4.13 PAPER TAPE SEGMENT ROUTINE ($PREEL)

A large subroutine library may require a volume of binary tape too great to be handled in one reel. The DBOS command, $PREEL, will punch a $REEL image to terminate a tape segment. The user may use this command to terminate any number of segments. See section 2.5.3 for use of $REEL.

2.5 PREDEFINED DBOS PROCESSORS

During system generation (section 4), a group of predefined DBOS processors are copied into the directoried core image program data file (DC). These, and other non-predefined processors, may be loaded and executed by use of the processor execution control command (subparagraph 2.4.11).

The standard predefined processors are:

a. Symbolic assembler.

b. FORTRAN compiler.

c. Core image converter.

d. Source image editor.

e. Debug program.

Each of these processors is described in the following subparagraphs.

2.5.1 SYMBOLIC ASSEMBLER

The GA 18/30 symbolic assembler is a two-pass assembler and is given the name A in the DC file. Thus, it can be loaded into core and executed by use of the control command:

```
$A
```

The assembler uses the following logical units:

SI	Source input (pass 1 input)
LO	Listing output
BO	Binary output
IS	Intermediate storage (pass 1 output, pass 2 input)

If the SI unit is a disk file, then the IS unit should be assigned to NO to avoid disk duplication during pass 1.

If the IS unit is assigned to NO, both passes are taken from the SI unit. Thus if the SI unit is the card reader and IS is assigned to NO the symbolic source deck must be input twice.

2.5.1.1 DBOS Assembler Extensions

Refer to the 18/30 Programming Operations Manual for assembler usage data. The DBOS assembler has been extended beyond the basic GA assembler. These extensions are enumerated in the following paragraphs.

ASCII Text (ASC) Pseudo-op

The ASC pseudo-op is identical to the EBC statement described in the GA 18/30 Programming/Operations Manual except that ASCII data strings are generated.

REF/DEF Pseudo-op

These features permit a program to REFerence symbols DEFined in other external programs. The term external is meant to indicate a program or storage location not assembled with the object program. A data table separately assembled but referenced by the object program would be an example.

DEF

A DEF pseudo-op is used to specify that the symbol in its variable field may be REFerenced by an external program. A DEF statement may not appear in an absolute program. All DEF's must appear at the beginning of the source file to which they make reference. A DEF is identical to an ENT except that the defined symbol need not be a program entry point. It is permissable to define a symbol used in the variable field of a DEF with an EQU statement, i.e.,

	DEF	OUT01	
OUT01	EQU	/57	Assign absolute value hex 57 to symbol
	END		OUT01. A REF to OUT01 will result in
			the value /57 in the variable field.

A maximum of 30 ENT and/or DEF symbols may be included in a single program.

REF

A REF pseudo-op specifies that the symbol in its variable field is external. REF's may occur anywhere in a program. Symbols which are declared as external by REF's may occur in a multiple item expression except as an operand of a multiply (*) operator. Machine instructions which contain REFed symbols in their variable fields must be of the two word or long format.

Conditional Assembly

A DO pseudo-op has been provided to permit a programmer to include/exclude selected source statements.

The statement:

```
        DO        L        M,N
```

directs the assembler to assemble the next M lines N times. The values must fall within the range of 0 and 255. All symbols used must be previously defined. If M, the number of lines, is greater than one, N must be zero or one. If N is omitted, it is assumed to be one. None of the statements within the range of the DO can be another DO.

Examples:

DO	L	3,1	Assemble the next 3 lines
DC		1	
DC		2	
DC		3	
DO	L	3,0	Do not assemble the next 3 lines
DC		1	
DC		2	
DC		3	

Source Data Preparation Format

The DBOS Assembler will accept source statements which originate in column 1 or 21. The remainder of the statement must be punched in relative columns, i.e., the OP code field is either started in column 7 or 27. A maximum of 60 columns of data is read and interpreted.

2.5.2 FORTRAN COMPILER

The GA 18/30 FORTRAN compiler is given the name F in the DC file. Thus it can be loaded into core and executed by use of the control command:

```
$F
```

FORTRAN uses the following logical units:

SI	Source input
LO	Listing output
BO	Binary output

FORTRAN is a one-pass compiler and requires no intermediate storage. The FORTRAN logical unit number u in the FORTRAN I/O statements (e.g., READ (u,f) list) will reference DBOS logical unit u. The user may use the standard assignment described in table 2.1 or define his own assignments with the executive command.

```
$u=file.
```

The *IOCS control card has no purpose and may be omitted. The user is advised to use the standard DBOS LUN assignments. (See table 2-1) This procedure permits any system LUN reassignments to be effective for all programs operating under the system.

Example:

if $lun=CR

then READ(lun,f) list would cause data to be input from the card reader.

Note: The FORTRAN disk READ and WRITE operations will always use logical unit 13 which must be assigned to file DK (standard assignment). The DK file is the only unpacked disk file. The disk limits for the DK file may be preset to any area of the disk. See section 2.8.1.

Refer to the IBM FORTRAN IV manual for compiler usage data (C26-3715-4).

Note: The * control cards which are used to specify compiler options are not listed.

Note: The *ONE WORD INTEGERS is the default option when not specified.

2.5.2.1 DBOS FORTRAN Extensions

Introduction

General Automation supplies subroutine library extensions for each of its executives and operating systems. These routines generally are supplied to permit access to Monitor functions by the FORTRAN programmer. This section will be updated as new routines are made available.

Array Characteristics

FORTRAN on the 18/30 stores arrays in reverse order. That is, ARRAY(1) refers to the highest core address assigned to the array. This arrangement is contrary to the manner in which the machine and Monitor store data. In the following extension discussion paragraphs special characters will be used. These characters are defined as:

N = length of array (number of variables)

IQ = N+1 value to be used as subscript base

FORTRAN Logical I/O Interface

This subroutine provides access to the logical I/O system of DBOS. The subroutine operates in two modes controlled by the first argument. Details of LIO are described in Section 5.0.

Mode 1 Data Transfer and Control

CALL FLIO(I,J(IQ-1))

where:

FLIO= name of routine

I = control variable for LIO>0

J = a dimensioned array. The first variable in array J must define the length of array J.

i.e., J(IQ-1) = length in words

J(IQ-2 through IQ-N) = Data

This call results in a call to LIO of the type:

```
CALL    LIO
DC      (I)
DC      J
DC      0
```

Mode 2 Device Status Test and Return

CALL FLIO(I,J)

where:

FLIO= name of routine

I = control variable for LIO and must be <0

J = a variable which will contain the device status word upon return from FLIO.

This call results in a call to LIO of the type

```
CALL    LIO
DC      (I)
STO     J
```

The use of FLIO requires that a control variable "I" be established. This variable can be defined in a DATA statement.

Example: READ 54 binary characters into array J from logical unit 12.

```
DIMENSION J(55)
DATA IRASC/Z110C/
J(55) = 54
CALL FLIO(IRASC,J(55))
```

Example: To test status of logical unit 12

```
DATA ITEST/ZF00C/
CALL FLIO(ITEST,K)
```

The status of logical unit 12 will be stored in variable K.

Note: A TEST operation must be performed before the next read/write call. LIO returns immediately to the user and does not wait for operation complete.

FORTRAN DBOS BULKN Interface

This subroutine provides access to the bulk handling routine (BULKN) in DBOS.

```
CALL BULK(I,J,K)
```

where:

BULK= name of routine

I = BULKN function (use DATA statement)

J = a dimensioned array such that

J(IQ-1) = word count

J(IQ-2) = sector address

J(IQ-3 through IQ-N) is data

K = variable which will contain error status on return.

BULK waits for operation complete status before returning to the user.

Refer to Section 6.4.2 for a detailed description of BULKN.

Extreme caution must be exercised when using this subroutine. It is possible to write anywhere on the disk including areas occupied by the Monitor and files. For safety, use FLIO or DEFINE FILE which monitors file boundaries.

2.5.3 CORE IMAGE CONVERTER

The GA 18/30 Core Image Converter is given the name CIC in the DC file. Thus it can be loaded into core and executed by use of the Control command:

```
$CIC
```

The Core Image Converter uses the following logical units to perform a core image file build:

BI	Primary binary input
SB	Secondary binary input (optional)
UL	User Library (optional)
LB	System library
CI	Binary output
IS	Intermediate storage (used to temporarily store object modules from BI and SB)
SL	Load map output (optional), missing subroutines list, and error messages
OM	Operator messages
CC	Control command input

The Core Image Converter (CIC) performs the functions of fixing relocatable object code, linking together main programs and subroutines and producing an absolute core image file which can then be loaded from the disk and executed by the DBOS disk loader. All CIC core image file builds are performed by making two passes over the object data. The first pass is required to build a list of referenced subroutines, resolve the subroutine entry addresses and obtain a map of memory. During the second pass, the executable core image output file is produced on logical unit CI (normally the WC file). Note that the CIC does not load the executable program directly into memory, hence all of available memory may be allocated and used during problem program execution.

The Core Image Converter can accept object program modules from up to four logical files. The CIC will first reference BI, which must contain the MAIN program as it's first object module and any number of subroutine modules. Optionally, subroutines may be input from [SB]. Both BI and SB must be terminated by $EOD image records (use $EOD command). All subroutines included in BI and SB are incorporated into CI whether actually referenced or not. The user may therefore include object modules which will be used in place of standard library routines to better satisfy his requirements even though they are not explicitly called out from his program logic.

The final two input files are subroutine libraries [UL] (optional) and LB. Only those subroutines actually called out in the load process will be included in CI. The CIC will make multiple passes over a library to satisfy references. This feature makes it unnecessary to 'level' a subroutine library. To optimize processing time the use of leveled libraries is preferred.

CIC Control Commands:

(Items enclosed in brackets are optional.)

```
[*MAP]
[*BOUND [,low][,high][,common][,INSKEL common]]
 *BUILD [,SB][,UL]
```

MAP - The optional MAP command provides a memory map of the resulting core image file.

BOUND - The optional BOUND command provides a means to override the default memory boundary values. These defaults are designed to maximize user core in batch job operation. The default value for each optional field is defined below. Any or all of these values may be specified, however all values between the BOUND and the particular value must be specified.

Parameter Definitions

low - first location to be occupied by program.

high - last available location for program.

common - highest address assigned for common data storage. Data is stored downward toward core location zero from address common.

INSKEL common - highest address assigned for INSKEL common data storage. Data is stored downward in core toward location zero from address INSKEL common.

Default Definitions for BOUND Parameters

If no BOUND command is specified or some fields are selectively omitted the following rules apply for determining default values.

low - the first location following the DBOS resident monitor.

high - the last location available to DBOS (usually end of core).

common - set to same value as high. (Note: COMMON data is stored backwards in core.)

INSKEL common - the origin of INSKEL common is defined by the expression (COMMON - size of common). (Note: INSKEL common data is stored backwards in core.)

BUILD - The BUILD command initiates core image conversion. It must be the last CIC command. [SB] and [UL] specify optional binary inputs. BI will always be the first logical file and LB will be the last logical file.

CIC Error Messages

Error Messages

CIC error messages are output on SL and prefixed by two slashes (//). Error messages discussed in this section always cause CIC to abort.

MESSAGE	MEANING AND REMEDIAL ACTION
//CHECKSUM ERROR	A particular card image element within the object module is either missing, out of sequence or was not punched properly when generated. Regenerate the object module.
//CODE ERROR	A card image contains a card type code not processed by the CIC. ILS and ISS subroutine header cards are not processed. Remove the object module from the core image build.
//PRECISION ERROR	A subroutine object module input from either BI or SB has a precision code specification different from that of the MAIN program. Change the precision definition of the object module or include the correct object module.
//END OF FILE	An end-of-file was encountered on either the IS or CI file. Allocate more storage for the working files. (see section 2.8.1)
//MISSING ROUTINES	This message is output followed by a list of the subroutines referenced but not included in any of the specified input files. Include the required object modules and restart the core image build.
//CC ERROR	A card image record which did not contain an asterisk in character position 1 was encountered before the BUILD directive. Include or correct CIC Control Commands.
//RANGE ERROR	The program being built is too large for the specified core area. The parameters specified on the BOUND statement may be changed if used.

CIC Operator Messages (for paper tape input only)

When the CIC encounters a $REEL record the following message is output on logical unit OM followed by a type-in request:

```
//REEL
? (input request)
```

2.5.4 SOURCE IMAGE EDITOR

The GA 18/30 source image editor is given the name EDIT in the DC file. Thus it can be loaded into core and executed by the use of the control command.

$EDIT

File Usage

The following files are used by EDIT.

SI	Source input
SO	Edited source output
CC	Control commands and insert lines
LO	Edited source list
SL	Control command list

Control Command

Control commands (identified by an "@" as the first character of a line) and Insert Lines to the source deck are all input from device CC. Blanks are not permitted in Control Commands. In the description below, optional elements of a control command are enclosed in braces.

A file to be edited is input via device SI. SI may be assigned to the card reader, a directoried source file (DS, NAME) or a user file. Format:

$SI=CR

$SI=DS (NAME)

$$\$SI=file \begin{bmatrix} \text{(name)} \\ \text{(bs-es)} \end{bmatrix}$$

Editing commands are input from device CC, usually the TTY keyboard. CC may be assigned to a device other than the teletype but may not be the device assigned to SI. Examples:

If SI = DS (NAME)

$$\text{or} = \text{file} \begin{bmatrix} \text{(name)} \\ \text{(bs-es)} \end{bmatrix} \quad \text{then}$$

$CC = CR is permissible.

Editor Control Commands

For the following, the character "Y" or the character "N" must follow the "=" character. The chara cter shown is the default case, which will be assumed upon entrance to EDIT.

@B=Y   Insert blanks

For B=Y, the generated source output will be preceded by 20 blanks, thus permitting the assembler format to be generated without need of spacing. Data already having the leading 20 blanks will be passed unmodified. For B = N, the records will be passed unmodified.

@O=N   Online mode

For O=Y, the On-Line mode will be entered, with the additional editing controls described below.

@L=N   List control

For L=Y, the full Edited Source Output will be listed while for L=N, only the changes are listed.

@S=Y Sequence control

For S=Y, the listing will be resequenced while for S=N, the listing will reflect the sequence number of the Source Input.

```
    Y
@C=
    N
```

For C=Y, the balance of the source input is copied to the output file. For C=N, the output file is terminated at its current position.

On Line Commands

When in the On Line mode, the following two additional commands may be used. In addition, the line reached by an L + n option or by the following two options, will be listed:

@- Delete the current line and advance to the next.

@+ Copy the current line and advance to the next.

Editing Commands

All editing commands except the Completion Command may be followed by one or more lines to be inserted at the current position of the input record.

@[L] [+n] [, m]

@	Indicates line is command
L	is a 1-5 character label.
n	is the count of lines after L has been reached. If L is omitted, n is an absolute line number with the first line of the program having n=1.
m	is the number of lines to be deleted. (Assumed to be zero if omitted.)

In operation, the source input is read and copied to the output file until line L + n is reached. Then if m is specified, m lines, including the current line, are deleted after which any inserts are input. If m is omitted, the line L + n is copied to the output after which any inserts are input.

Editing examples:

Assume a freshly key punched deck is to be processed. The operator wishes to list his deck, with sequence numbers, make corrections and then assemble his program.

$JOB	
$COPY,CR,DS(NAME)	Place source on disk
$SI=DS(NAME)	Editor source input from new file
$EDIT	Call editor
@L=Y	List all
@S=Y	Provide line numbers
@C=Y	Copy DS(NAME) to WS to produce a sequenced listing.

At this point the operator may punch an edit deck or edit from the teletype.

```
$CC=TY or $CC=CR
```

Editing procedure examples:

To delete a line at label +3 the edit command

```
@Label+3,1
```

To delete n lines at starting at line number 2384 the edit command

```
@+2384,n
```

To add a line @ line 200 with no delete the edit command

@+200

data to be inserted, may include leading blanks.

NOTE

No leading blanks are required because "B" edit command was = Yes.

To delete m lines starting at label + n and then make insertions the edit sequence would be:

```
@label+n,m
data records
   .
   .
   .
data records
etc . . .
```

In all cases the master file DS(NAME) is copied to WS until the to be edited line is encountered. At this point the edit functions specified are performed.

The last record input from device CC must be an $@C=^{Y}_{N}$ command.

This operation will complete the copying of the source file (DS,(NAME)) to WS.

A new sequence listing was produced during the edit operation. The operator may alter the list status any time by @ L=$\genfrac{}{}{0pt}{}{Y}{N}$. At this point the user may assemble his program to determine any additional source errors. This may be accomplished as follows:

$SI=WS	assembler source from edited file
$IS=NO	No intermediate storage required as source is already on disk
[$BO=NO]	No binary output.
$A	Call assembler

The operator may now edit his source file further. If editing was accomplished from card data add the necessary new edit cards. Call the editor as above.

If editing was done from the TTY two alternatives are available.

1. Reenter all edit commands plus the new ones via the keyboard.
2. Copy the partially edited file back to DS(NAME). The listing produced during the previous edit would be used for reference.

Example:

$REPLACE,DS(NAME),WS	Edited file to DS
$PACK,DS	Compress disk file
$EDIT	Start next edit.

The card edit procedure is preferred. This method maintains one master file (that which was originally copied) and an edit deck. Recovery in the event of mishap is positive and simplified.

Once a source file has been totally edited, it should replace the unedited file. Refer to example above. The PACK operation closes up the disk file by removing the old source data.

2.5.5 DEBUG PROGRAM

The GA 18/30 Debug program is given the name D in the DC file. Thus, it can be loaded into core and executed by use of the command

$D

Typically, however, the DEBUG program is loaded in conjunction with a program to be debugged:

$LOAD,D	Load the program in the WC file and load D. Execute D.
$pname,D	Load the program given the name 'pname' in the DC file and load D. Execute D.

DEBUG uses the following logical units:

CC	command input
OM	listing output, error message output, secondary command input
LO	listing output, command output, error message output
CI	binary output

Debug occupies approximately /700 locations of high core origined at /7900. Programs to be debugged must not occupy storage above /7900 (truncated to to reflect available core).

COMMAND FORMAT

A command consists of a one-letter operator and one or more operands. Multiple commands may be placed on a line. The slash (/) delimits commands; the comma (,) delimits operands. No comma should be placed between the operator and the first operand. A space terminates the last command, optional commentary may follow (for teletype input, a carriage return terminates the command line; no space is required unless the user includes optional commentary.) Example:

Command Operand,Operand . . . ,Operand/Command,Oper, . . .

OPERAND FORMATS

The following are the possible DEBUG operands:

1. Addresses/Constants

   An address/constant consists of 1 - 4 hexidecimal digits, written as /nnnn, with an optional modifier of R,X,L,P or S. If more than four digits are input, only the last four are considered; non-hexadecimal characters (other than modifiers) are ignored. The modifier may be placed anywhere in the digit sequence and has the following effect on an address (not applicable to constants):

None	Memory address
R	Register buffer address
X	Disk buffer address
L	Limit buffer address
P	Program relative address
S	Sector address

   The S modifier is not allowed for certain operands.

Examples of operand addresses are:

3A location /3A or constant /3A

R3 register 3

31X location /31 of disk buffer

AS6 sector address /A6

P43 program relative address /43

2. Address/Constant Field

An address/constant field is of the form:

a. A pair of addresses/constants separated by a dash. The modifier of the second address is ignored (and may be omitted, it is assumed to use the same modifier as the first address).

b. A single address; may be used when the first and last address are the same.

c. The letters R,X,L and P are fields which are defined as follows:

R = R0-6

X = X0-13F

L = (contents defined by user) L0-1

P = contents of L0-contents of L1

(Any reference to the letters R,X,L or P without an address constant is interpreted as a reference to the entire field.)

In the text which follows the lower case letters a,ca,rf,f,k,kf denote:

a	address or constant, S modifier allowed
ca	core address, no S modifier
f	address or constant field
cf	core field, no S modifier
k	a 1-4 digit hexadecimal constant
kf	double word constant

Refer to table A.

BUFFERS

Four buffers are accessible to the user.

1. Register buffer. When control passes from the user program to the DEBUG program, the register buffer is set as follows:

R0	Instruction counter
R1	Index register 1
R2	Index register 2
R3	Index register 3
R4	A register
R5	Q register
R6	Carry and overflow (bits 14 and 15)

2. Disk buffer. This buffer holds one disk sector (320 words) which can be used as a work area for disk modifications.

3. Limit buffer and Program buffer. The limit buffer (2 words) defines the limits of the program buffer. (L0) is the low program address and (L1) is

the high program address. Proper definition of the limit buffer allows relative references to relocatable program addresses. The limit buffer is normally defined by the replace command; e.g.,

RL,1092,2301 Low program buffer limit = 1092, high=2301

then P0=/1092 and P (buffer)=P0-126F=/1092-/2301

A reference to P100 would give the value of relative location 1192. A reference to P by itself would refer to a field defined as P0-126F.

DEBUG COMMANDS

Table A indicates allowable DEBUG commands and their function. Items in brackets are optional. The notation (...) indicates that the operand sequence may be repeated. A more complete explanation of each command, along with examples, is given below.

1. Type 'T'

   $Tf_1, f_2 \ldots, f_n$

   The Type command causes the contents of each location within a field to be output to OM. Any number of fields may be specified following the type command. After output of all fields is complete, the user has 3 options:

   a. Enter a new command.

   b. Enter a carriage return (blank record for card input). The successive location and its contents will be output, i.e., if P52 was just output, P53 would be output.

   c. Enter a hexadecimal constant. This constant will replace the contents of the last location output; then the successive location and its contents will be output.

   Options 'b' and 'c' are available only when:

   a. The type command is the last command of a sequence e.g.,

      P37/T36,47,R3 (Print location 37, Type location 36, 47 and the contents of R3.)

At this point R3 and successive locations can be modified.

Options 'b' and 'c' are not available in the example below

T36,47-A1/P52

P refers to a command and is recognized as last command because slash delimiter was processed.

b. In the following example the letter 'P' denotes a program location and may, therefore, be modified.

TS3,P4 (Type contents of disk sector 3, and relative program location 4.)

An exception to the last item may be changed option exists if the last item referred to was a disk location. Example: TP4,X30,R3,S25 (S25 refers to disk sector 25.)

2. PRINT 'P'

$Pf_1,f_2 \cdots ,f_n$

The print command causes the contents of each location within the field to be output to LO. No subsequent modification is allowed. An example is:

P30,R,S306,500-800,X4-9,P

which will print absolute core location /30, the entire register buffer, disk sector /306, core locations /500-/800, locations /4-/9 of the disk buffer, and all of the program buffer. To print program buffer location 30 the command PP30 is required.

3. REPLACE

$Rf,k_1,k_2...,k_n$

This command replaces the contents of 'f' with the pattern $,k_1$ $,k_2$ ... $,k_n$. If the pattern is shorter than the field length, the pattern is repeated until the field is exhausted. (If 'f' is a sector field, the pattern is expanded into the disk buffer (320 words) and the buffer is output to each sector in the field.) If no pattern is supplied, a pattern of 0 is assumed. An alternate form of the replace command is:

$Rca,k_1,k_2...,k_n$

In this form, one copy of the pattern replaces the contents of the locations starting at 'ca'. Examples of the replace command are:

R61	Zero location /61
RX	Zero the disk buffer
RS90-103	Zero sectors /90-/103
R80,4	Replace location /80 with /4
RL,1057,1A32	Define the limit buffer
RR0,3	Replace register 0 with 3.
RX,1,24,8	Fill the disk buffer with the repeating pattern /1,/24,/8.
R83,A2,B7,C4	Replace /83-/85 with /A2,/B7,/C4.

4. MOVE 'M'

$Mp_1, p_2 \ldots, p_n$ $p=f_1, f_2$ f_1=sending field.

Move f_1 to f_2. If the sending and receiving fields are both core or both disk, the receiving field determines the number of elements to move. If one field is core and the other field is disk, the core field determines the number of elements to move. Data movement starts from the low address. Examples of the move instruction are:

M70,80-A0	Move locations /70-/90 to /80-/A0.
M70-500,80-A0	Move locations /70-/90 to /80-/A0. (receiving field limit control)
MP,S501	Move the program buffer to disk starting at sector /501.
MS501,P	Move disk starting at sector /501 to the program buffer. Preset limits with RL, l_1, l_2
MS10,S11-20	Copy sector /10 to sectors /11 to /20.

Note:
The command MS5,S11-20 will copy disk sectors starting at sector 5 to sector 11,6 to 12,7-13 etc.

The command MS10,S11-20 above copies sector 10 into each sector from sector 11 to 20. The key point here is that the source sector (or core location if moving core), is only one field length below the destination area. Each successive move or copy effectively copies the same information.

5. EXECUTE/TRAP 'X'

X [ca] [,k] [/c_1] [/c_2] . . . [/c_N]

The X command format provides the operator with program entry address selection, trap address selection, trap loop count and execution control. The X command initiates program execution by placing the contents of registers R0-6 into their hardware counterparts. The value of R0 replaces the instruction counter and is therefore the point of execution.

To specify an execution address:

RR0,a	Replace register R0 with execution address
X	Execute

The X command may be written as part of the replace sequence.

RR0,a/X

To set up a trap condition the X command is followed by one or two parameters.

X,ca,k

Execute from core location specified by contents of R0. TRAP at location 'ca' after 'k' encounters of trap location. If 'k' is not specified one is assumed.

After a TRAP is encountered register R0 is equal to the trap address. A subsequent X command will resume execution from the TRAP location (ca).

A series of Debug commands may be specified to be executed each time the TRAP is encountered.

$X,ca,k/c_1/c_2.../c_n$

After TRAP at location 'ca' execute specified Debug commands $c_1,c_2...c_n$

The command sequence is executed before the trapped instructions are executed. When control returns to DEBUG, R0 is defined as the trap address, the trap locations are restored, but have not been executed. The following rules must be followed in selecting a trap location:

a. The contents of ca and ca+1 must not be modified or referenced during program execution. Control must pass to ca, not ca+1.

b. If the trap execution count is greater than one, the trapped locations may not be instruction counter relative (the instructions are not executed in place). Also, the command sequence may not contain an 'X' operator if k is greater than one.

After an object program and debug have been loaded ($NAME,D or $LOAD,D) R0 is set to the entry point of the loaded program. Execution may be started immediately by entering a X command to debug.

Examples of the Execute/Trap command are:

X	Execute at R0.
XA53	Set trap at /A53, execute at R0. Return to DEBUG when the trap is encountered.
X62,5	Set trap at /62, execute at R0. Return to DEBUG, after the fifth encounter of the trap.
XA32,A/PR/T54	Set trap at /A32, execute at R0. After each encounter of the trap, print the register buffer and type location /54. Return to DEBUG after the trap is encountered /A times.
X104/PR/X32/T80,R3	Set trap at /104, execute at R0. When the trap is encountered, print the register buffer. Set a trap at /32, execute at R0 (=104). When the trap is encountered, type location /80 and register 3.

6. SEARCH 'S'

Scf,kf_1[,kf_2]   Search for kf_1.

The search command searches cf for kf_1. Each match is output to OM. If kf_2 (a mask) is supplied, kf_1 is compared with the logical product of cf and kf_2, element by element. kf_1 is a single or double word constant. A doubleword constant is of the form k_1-k_2 where k_1 is the high order word. Similarly, kf_2 may be a single or doubleword mask.

If kf_1 is a doubleword and kf_2 is not, then kf_2 is assumed to be of the form k_1-FFFF. Examples of the search command are:

S100-500,ABCD	Search locations /100 to /500 for /ABCD
SP,3781-46A2	Search the program buffer for the doubleword /3781,/46A2.
SX,157,FFF	Search the disk buffer for /157. Mask each element with /FFF.

7. OBJECT OUTPUT 'O'

$Ocf_1 \left[,cf_2\right] \ldots \left[,cf_n\right] \left[,k\right]$ Object output to CI.

Output the core fields in 54 word binary object format to CI. If a core field represents a single address, it must be of the form a_1-a_1, otherwise the address will be interpreted as the execution address (k). The execution address may appear anywhere in the operand sequence. If more than one execution address appears, only the last is considered. If the execution address is omitted, R0 is assumed. The DEBUG buffers may be included as operands, e.g.,

O1000-1500,R,X

or if L is defined as l=1000 and e=1500, then

OP,L,R,X

At a future time the $LOAD,D will restore the user program and DEBUG buffers and debugging may continue.

This feature permits partially debugged program segments to be saved, should the program be destroyed during debug of the next segment a simple reload of the patched program is possible. The program may be stored as a directoried file also. ($COPY,WC,DC(NAME)) In this case $NAME,D would be used to reload.

8. SWITCH COMMAND INPUT. 'K'

   The K command switches command input from CC to OM or OM to CC. This command allows program modification commands to be punched on cards or paper tape; by using K as the last command, control will pass to OM (normally the teletype) to allow further modifications.

9. RETURN TO DBOS. 'Z'

   The Z command terminates the DEBUG program by returning to DBOS.

OUTPUT FORMAT

A DEBUG listed line has the format

$loc=data_1\ data_2 ... data_n$

where:

loc	represents the location of the first data item in the line. loc is of the form
hhh	core address
Rh	register address
Xhhh	disk buffer address
Lh	limit buffer address
Phhh	program relative address (if the relative address exceeds /FFF the modifier increments alphabetically to Q,R,S,etc.
$data_i$	represents a data item (hexadecimal)
n	is 16 for LO and 8 for OM.

Table A

$Tf_1[,f_2]...[,f_n]$	Output to OM, allow changes
$Pf_1[,f_2]...[,f_n]$	Output to LP
$Rf[,k_1][,k_2]...[,k_n]$	Replace f with $[,k_1][,k_2]...[,k_n]$
$Mp_1[,p_2]...[,p_n]$ $p = f_1,f_2$	Move f_1 to f_2.
$X[ca][,k][/C_1][/C_2]...[/C_n]$	Execute until trap (C=command)
$Scf,kf_1[,kf_2]$	Search f for kf_1 with mask kf_2
$Ocf_1[,cf_2]...[,cf_n][,k]$	Object output, k=execution address
K	Change command input
Z	Return to DBOS.

a	Address/constant. address may have modifiers of R,X,L,P or S.
ca	Core address. Same as 'a', but S modifier not allowed
k	A constant
f	Address/constant field of the form $a_1 - a_2$, a, or R,X,L,P
cf	Core field. Same as 'f', but S modifier not allowed.
kf	Constant field of the form k or k_1-k_2 (doubleword constant)

2.5.6 SEQUENCE/COMPARE PROGRAM

The Sequence and Compare utility program permits the user to copy, and sequence and compare (verify) files. The program uses LIO and may therefore copy from and to any file.

The GA 18/30 Sequence/Compare utility is given the name SQCM in the DC file. Thus, it can be loaded by use of the command:

```
$SQCM
```

The command format is:

```
$SQCM
>[base] [,increment] [,mode]
```

where:

>	Indicates a SQCM command statement.
[base]	Is n alphanumeric characters (except blanks). These characters will replace the n rightmost characters of the input (SI) file image. Blanks may be included by use of the # symbol. Typically, the last few characters of 'base' are numeric establishing the first sequence number. Note: A limit of 8 characters is imposed if the mode specified is (B or U). If no base is specified, SQCM will perform a straight copy or compare.

[, increment] A positive value of any size which SQCM will use to increment the base. If no increment is specified, SQCM will use the numeric portion of 'base' as the increment.

[, mode] Input data mode specification. Four modes are accepted:

A - 40 word ASCII

B - 54 word binary

C - 60 word binary

U - unformatted paper tape

If mode is omitted, ASCII is assumed.

If 'U' is specified and the input device is not 'PR', 'C' input is assumed. The output mode is the same as the input mode except for 'B' input, which is output in 'C' mode. If 'U' is specified and the output device is not 'PP', 'C' output is assumed.

SQCM uses the following logical files:

CC - command input

OM - operator I/O

SI - data input

SO - data output

15 - compare/copy switch

Copy/Compare Control

If \$15=NO, SQCM will copy the file from SI modified by sequence data if specified to SO. If \$15≠NO, then SQCM will compare the file input from SI modified by sequence data if specified to file assigned to \$15.

Error Detection and Correction

On compare error, the correct image will be output to SO and the message ERROR will be output to OM. The correct image will be taken from SI modified by sequence information. The operator must respond after an error with one of the following:

C - Continue and ignore error.

R - Reread image to double check or verify new image output when error occurred.

D - Terminate and return to the DBOS monitor.

End of File Control

Each data mode has a unique end-of-file condition. Operation will terminate when an end-of-file is encountered. They are:

A - Any number of leading blanks followed by the characters "END".

Note: The word END may not start in column 1 or 21 as it will be ignored.

B - A type 'F' object record.

C - The characters $EOD in columns 1-4 of a card image.

U - 15 inches of blank trailer (blank leader is ignored).

Note: The end-of-file image is both sequenced and compared as part of the file.

Control Command Insertions

Any number of ASCII records may precede the SQCM command (>). These records will be copied to or compared with the object file. The inserted records are not sequenced or titled. This feature permits annotation or control commands to be inserted prior to the object data.

A special termination procedure is provided if only insertions are desired.

Example:

```
SQCM
Insert 1
Insert 2
>>                  Terminate job after Insertion number 2.
```

Examples of SQCM Operation

SAMPLES OF SQCM USAGE

1. ASSUME A BINARY FILE IS ALREADY IN WC FROM CIC

TO GENERATE A SEQUENCED BACKUP OBJECT DECK DO

```
$JOB
$SI=WC                 INPUT FROM WC
$SO=CP                 OUTPUT TO CP
$SQCM                  CALL SQCM
>NAM001,B              NAME PROGRAM NAM,  SEQ FROM 1 BY 1S
```

TO VERIFY DECK

COMPARE INPUT FROM CARD READER
CALL SQCM
SUPPLY SAME SEQUENCE CONTROL AS ABOVE
CORRECTIONS, IF ANY, WILL BE OUTPUT TO CP

```
$15=CR
$SQCM
>NAM001,B
```

2. TO SEQUENCE AN ASSEMBLER OR FORTRAN SOURCE FILE INTO CARDS AND VERIFY.

```
$JOB
$COPY,CR,WS            INPUT UNSEQUENCED DECK FROM CARDS TO WS.
$SI=WS                 SOURCE FROM WS
$SO=LP                 VERIFY FILE, IF DESIRED, AND LIST ERRORS ON LP.
$15=CR
$SQCM                  VERIFY SOURCE FILE DISK IMAGE
>,A
$SO=CP                 SEQUENCE OUTPUT TO CP
$15=NO
$SQCM                  CALL SQCM
>TITLE00000,10,A       FIVE CHARACTER TITLE, FIVE DIGIT SEQ. NO., INC.X10
                       INPUT WILL TERMINATE ON SOURCE FILE 'END' STATEMENT
$15=CR
$SQCM                  CALL SQCM
>TITLE00000,10,A       COMPARE DECK TO FILE ON DISK MODIFIED BY SEQ DATA
```

3. TO PRODUCE A TITLED AND SEQUENCED CARD DECK AFTER A FORTRAN COMPILATION AND BUILD.

```
$JOB
$F                     COMPILE MAINLINE
      SOURCE STATEMENTS FOR FORTRAN MAINLINE
$A                     ASSM SUB-PROGRAM
      SOURCE STATEMENTS FOR OPTIONAL SUB-PROGRAM
      SOURCE STATEMENTS, MUST TERMINATE WITH 'END' STATEMENT
$EOD                   CLOSE WB FILE
$CIC                   BUILD CORE IMAGE FILE
*MAP
*BUILD
$SI=WC
$SO=CP
$SQCM                  SEQUENCE WC FILE TO CP AND TITLE AS FOR
>FOR00000,10,B
$15=CR
$SQCM                  COMPARE BACKUP DECK, CORRECTIONS TO CP
>FOR00000,10,B
```

4. TO ASSEMBLE A PROGRAM AND PRODUCE A SEQUENCED BACKUP CARD DECK WITH TITLE AND VERIFY.

```
$JOB
$A
      SOURCE STATEMENTS FOR ASSEMBLY,  MUST TERMINATE WITH AND 'END'
$SI=WB                INPUT FROM WB WHICH IS ASM OUTPUT
$SO=CP                SEQUENCED OUTPUT TO CP
                      SEQ SW.
$SQCM                 CALL SQCM
>TITL0001,B           SUPPLY SEQ AND TITLE DATA + FORMAT
$15=CR                SET COMPARE FROM CR
$SQCM                 CALL SQCM
>TITL0001,B           SEQ AND TITLE DATA + FORMAT
```

5. TO SEQUENCE A SOURCE FILE CONSISTING OF MULTIPLE STACKED JOBS WITH MANY 'END' STATEMENTS TO THE CP.

```
$JOB
$SI=CR                UNSEQUENCED SOURCE FROM CR
$SO=WS                SEQUENCED SOURCE TO WS
$15=NO                SEQUENCE/COPY SWITCH
$SQCM                 CALL SQCM
>TITL00001,C          TITLE + INCR OF/STARTING FROM 00001
     SOURCE FILE      ANY DECK OF CARDS NOT CONTAINING $EOD
$EOD                  TERMINATE FILE AS TYPE C UNFORMATTED DATA

$COPY,C,WS,CP         PUNCH CARD DECK FROM SEQUENCED FILE, INCLUDES $EOD

$SI=WS                SQCM COMPARE MASTER
$SO=CP                SQCM CORRECTIONS TO CP
$15=CR                SQCM COMPARE FILE
$SQCM                 CALL SQCM
                      COMPARE AS C FORMAT DATA
```

SEQUENCED SOURCE FILE WHICH INCLUDES '$EOD' RECORD

NOTE: THE SEQUENCED FILE IN WS IS NOT LISTABLE. THE SEQUENCED CARD DECK MAY BE LISTED WITH A SERIES OF COPY COMMANDS, ONE FOR EACH JOB. THE DECK MAY BE STORED AS A SERIES OF JOBS IN DS.

NOTE: THAT WHILE THE WHOLE FILE HAS A CONTINUOUS SEQUENCE NO. IT STILL CONSISTS OF MANY INDIVIDUAL FILES. THE FILE MAY BE COPIED IN C FORMAT BUT MAY NOT BE STORED IN DS IN C FORMAT. A SERIES OF COPY COMMANDS IS REQUIRED TO STORE THE FILE IN A FORMAT SUITABLE FOR LISTING OR INPUT TO THE PROCESSORS.

```
$COPY,C,WC,C,DC(TITL)                COPY DATA AS C FORMAT DATA
$COPY,C,WC,CP                        COPY C FORMAT DATA TO PUNCH
```

6. TO ASSEMBLE A PROGRAM, PRODUCE A SEQUENCED BACKUP CARD DECK WITH TITLES AND VERIFY + STORE IN DC FILE

```
$JOB
$A
    SOURCE STATEMENTS, LAST RECORD MUST BE AN 'END' CARD.

$SI=WB                    SQCM INPUT FROM WB
$SO=WC                    OUTPUT TO WC
$15=NO                    SEQ./COPY
$SQCM                     CALL SQCM
>TITL0000,1,B             TITL AND SEQ., INC =1, BINARY
$COPY,C,WC,C,DC(TITL)  STORE SEQUENCED IMAGE IN DC FILE.
$LDIR,DC
$C
$15=DC(TITL)                 COMPARE DC FILE TO WB
$SQCM
>TITL0000,1,B
$C
$COPY,C,WC,CP                OUTPUT SEQUENCED BINARY BACKUP DECK
$15=CR                    COMPARE TO CR
$SQCM                     CALL SQCM
>TITL0000,1,B             DO COMPARE
```

7. TO COPY AND VERIFY A PAPER TAPE OF ANY FORMAT

```
$JOB
$SI=PR                    FROM PR
$SO=WC                    TO   WC
$SQCM                     COPY UNFORMATTED DATA TO DISK
>,U
$15=WC
$SQCM                     VERIFY DISK FILE
>,U
$15=NO
$SO=PP
$SI=WC
$SQCM                     COPY WC FILE TO PP
>,U
$15=PR
$SQCM                     VERIFY PUNCHED TAPE
>,U
```

THE DISK IMAGE IN WC CAN BE STORED AS A DIRECTORIED FILE IN DC AS SHOWN BELOW. THE DATA IS NOT EXECUTABLE IN THIS FORM HOWEVER. THIS PROVISION IS FOR STORAGE ONLY

```
$COPY,C,WC,C,DC(NAME)     UNFORMATTED TAPE TO DISK FILE
```

TO RECOVER THIS FILE AND PUNCH A PAPER TAPE

```
$JOB                      TO PUNCH A BACKUP BINARY PAPER TAPE AND VERIFY
$SI=DC(NAME)
$SO=PP
$SQCM                     PUNCH NEW TAPE FROM DC FILE (NAME)
>,U
$15=PR
$SQCM                     VERIFY NEW TAPE, IF AN ERROR OCCURS THE TTY WILL
>,U                       TYPE 'ERROR'.  RETURN TO MONITOR (ENTER D C/R) AND
$C                        RETRY VERIFY OR PUNCH NEW TAPE.
```

2.5.7 SYSTEM GENERATION UTILITY

The System Generation Utility is designed primarily for initial system configurating and building as described in Section 4.0. SYSGN, however, is also useful as a utility routine. Used as a utility many system characteristics may be redefined. All redefined parameters become effective immediately upon return to the Monitor.

The GA System Generation Utility is given the name SYSGN in the DC file. Thus, it can be loaded via the command:

$SYSGN

The SYSGN utility inputs commands expressly designed to ease the generation procedure. The Debug feature is included in SYSGN and is described in Section 2.5.5.

System Generation Commands

All commands are written in standard DBOS format as described in Section 2.2.

Disk Unit Assignment

$DISK,unit

where unit is one of the following:

1341 - Model 1341 Disk Storage Unit

1344 - Model 1344 Disk Storage Unit

1345 - Model 1345 Disk Storage Unit

This command establishes the proper code in BULKN to handle the specified type disk. This command would not ordinarily be used in the utility mode.

Core Size Assignment

$CORE, size

where size is one of the following:

8K - for 8192 word memory

16K - for 16384 word memory

32K - for 32768 word memory

This command establishes the upper core address limit for many programs which execute under DBOS. The input value must reflect the actual available core in the system.

Disk File Limit Assignment

$file(bs-es)

where:

file	Is the name of a disk file as shown in table 2-2.
(bs-es)	Defines an area of contiguous sectors to be allocated to the named file. The values bs and es are specified in hexa-decimal.

This command is especially valuable when a currently defined file structure is found to be inadequate. Caution must be exercised when redefining files to not overlay a previously defined file. Typically, all files which follow the file to be redefined must also be redefined if they are to remain contiguous. A file may be redefined to follow the last previously defined file; however, the space vacated is frequently wasted. Sectors O-FF are reserved for system use. The ordering of disk files shown in appendix A has been selected to minimize seek time. It is preferable to reassign all files following a specific file to preserve the indicated order. All moved files must be rebuilt.

Initialize Disk

$IDISK [,bs-es]

where bs-es, if specified, are the beginning and ending sectors (in hexadecimal) to be initialized. If the optional limits are omitted, the entire disk is initialized.

Those sectors which are initialized will be cleared of all previous data and addressed. Each sector is verified for reading accuracy.

An accuracy failure in a sector marks that sector as a "bad Sector". The bad sector and its alternate are logged in the bad sector table, locations /66 to /6F of the Monitor. The bad sector table may be examined by use of the Debug routine. The bad sector table consists of word pairs which contain the bad sector and its alternate. A value of /FFFF,/FFFF is used for unused word pairs .

Define Initial and Standard Logical File Assignments

$lun=file [(bs-es)] ,P

This command is the same as described in Section 2.4.1 except that in SYSGN, the system basic definitions are established. The define file command in SYSGN is used to establish the disk resident lun/file assignment table. The table contains two assignments for each lun. The first assignment is in effect immediately after an initial system load from disk (IPL). The second or Standard assignment goes into effect after a JOB command is processed.

Procedurally, all Standard assignments must be made prior to the initial assignments. The command to define a Standard assignment also defines the initial assignment. Generally, the same file assignment is satisfactory for both the initial and Standard condition. In these cases, only one file assignment command is necessary.

Example:

$SL=LP,P	Standard and initial assignment is to line printer.

To force a special initial file assignment for a lun

$SL=TY	Initial assignment is TY which overrides previous assignment of LP.

Copy System Executive to Disk

$EXEC,file [,B],file$_2$(bs-es)

This command is a special form of the copy command used to write a new Executive to the disk. The parameter file$_2$ must be a disk file, usually defined as sectors /12-/27.

Example:

$EXEC,CR,B,DP(12-27)	Cards to disk
$EXEC,PR,B,DP(12-27)	Paper tape to disk

Initialize Directoried File

$IDIR,file

where file is a directoried disk file.

The purpose of this command is clear a directoried file, i.e., DC,DS,LB,UL. This command is used when it is desired to completely replace the contents of a file. The alternative to this command is to use the DELETE or REPLACE commands for all entries in the file.

Patch Monitor

$D

This command invokes the Debug routine described in section 2.5.5. Its purpose in SYSGN is to permit the user to examine and/or change the Monitor. The 'Z' command which normally returns to the Monitor will return control to SYSGN.

Write Monitor to Disk

$WMON

This command writes the system bootstrap and monitor to disk. This command must be executed prior to returning to DBOS.

Initial System Operation

$START

This command transfers control to DBOS. All modifications made with any of the SYSGN commands will be in effect immediately.

2.5.8 BOOTSTRAP LOADER GENERATOR

The Bootstrap Loader Generator is used to generate a card or paper tape bootstrap loader for stand alone use. The operator may precede the generation of an object deck or tape with this routine. This feature is especially advantageous to paper tape users. This loader recognizes type codes of O, A and F only. The assembler, Debug and core image converter outputs may be loaded.

The Bootstrap Loader Generator is given the name BOOT in the DC file. Thus, it can be loaded by use of the command:

$BOOT

BOOT uses the following logical units:

S0 - data output

15 - generate/compare input switch

OM - error messages

CC - operator input

Generate/Compare Control

If $15=NO, BOOT will output an IPL format bootstrap loader to SO. SO may be assigned to PP or CP. If $15≠NO, then BOOT will compare the input file assigned to SI with the correct bootstrap image.

Error Detection and Correction

On compare error, the correct image is output to SO and ERROR is output to OM. The operator must respond after an error with one of the following:

C - continue

R - reread image to double check or verify new image output when error occurred.

D - terminate and return to the DBOS monitor.

2.5.8.1 Card Bootstrap Loader Execution

The Card Bootstrap Loader may be loaded anywhere in core. Once loaded the loader may be re-entered at location Q as required for loading subsequent program modules. Care must be exercised to place the loader in an area of core which will not be overlaid by the object program. The execution/loading address (Q) must be an even address.

The following steps are required to load and execute the Card Bootstrap Loader at location (Q):

1. IPL bootstrap into location Q (normally zero). Q must be even
   a. Unlock the 'Console Enable' and 'WSPB' switches, set all switches off (up) except for 'SPO', 'IDLE', 'HALT', and card IPL channel (normally "Register Select' switches 8 and 4), press and release 'reset'.

b. Load card reader with 3 card bootstrap/loader followed by one or more object decks. Ready the card reader.

c. Set Q into the console data switches, press and release 'Enter', set 'Halt' switch off (up), press and release 'IPL' (card 1 of the bootstrap/loader will be input), set 'Halt' on (down).

2. Execute bootstrap at location Q with data switch options.

a. Press and release 'Reset', set Q into the console data switches, set register select switches off (up), press and release 'Enter'.

b. Select data switch options

Switch 0 = load/execute option

Off = Load according to switch 1

On = Execute last complete object deck loaded.

Switch 1 = load option

Off = Load and execute one object deck

On = Load multiple object decks

Both switches may be changed at any time during the execution of the loader, i.e., if multiple object decks are being loaded, switch 1 may be reset while loading the last object deck to effect execution, or switch 0 may be set after the last card in the reader is input to effect execution.

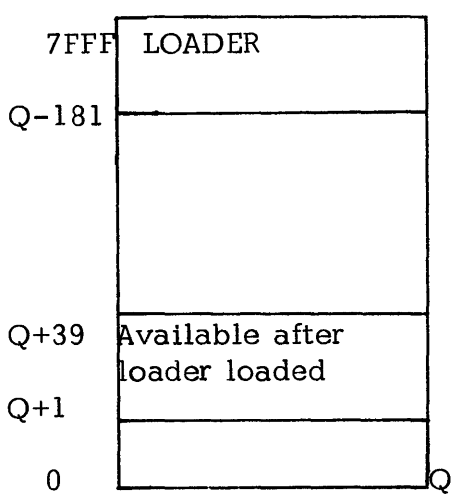
c. Set 'Idle/run' switch to 'Run', press and release 'Step' to execute the bootstrap/loader.

3. Errors

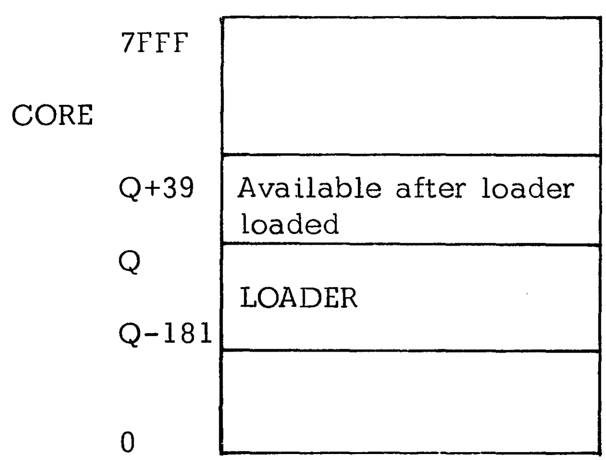
a. Checksum error or card reader error. Program will enter a wait condition. Correct error and ready reader to reinput card. Press and release step.

b. Loader overlay error. Occurs when object program destroys loader while loading. Choose a suitable Q (see core requirements) and return to 1A.

4. Core requirements. The Bootstrap (first card) requires 40 locations starting at Q. After the bootstrap has loaded the loader, locations Q+1 to Q+39 are available to the user. The loader (cards 2 and 3) requires 181 locations, the last of which is Q, the entry point. If Q is zero (normal case) the loader will reside in the top of core with Q at location zero.



Q = 0



Q = 1000_{16}

2.5.8.2 Paper Tape Bootstrap Loader Execution

The paper tape bootstrap loader may be loaded anywhere in core. Once loaded the loader may be re-entered at location Q as required for loading subsequent program modules. Care must be exercised to place the loader in an area of core which will not be occupied by the object program. The loading/execution address (Q) must be an even address.

The following steps are required to load and execute the Paper Tape Bootstrap Loader at location Q:

1. IPL or use teletype boot (programming/operations manual, 88A00121A, appendix G) to read loader into location Q. Q must be even. The IPL procedure is described below.
   a. Unlock the 'console enable' and 'WSPB' switches, set all switches off (up) except for 'SPO', 'idle', 'halt', and paper tape IPL channel (normally 'register select' switches 8 and 4), press and release 'reset'.
   b. Load paper tape reader with 'rubouts' over read head. Ready the reader.
   c. Set Q into the console data switches, press and release 'enter' set 'halt' switch off (up), press and release 'IPL' the entire loader should be read stopping on the terminating '5' punch. Set 'halt' down.

2. Execute loader at location Q.

   a. Press and release 'reset', set Q into the console data switches, set register select switches off (up). Press and release 'enter'.

   b. Load object paper tape into paper tape reader/teletype. Set 'idle/run' switch to 'run', press and release 'step' to execute the loader. If no errors occur, loading will stop after an end record (type 'F') is encountered. The A register will be zero. See data switch options.

3. Errors

   a. Checksum error. The A register will be non-zero.

   b. Loader overlay error. Occurs when object program destroys loader while loading. Choose a suitable Q (see core requirements).

4. Data switch options. Switch zero is used as follows:

   Off = load another tape, return to step 2b.

   On = execute last successfully loaded program. Press step.

5. Core requirements. The loader requires 176 locations starting at Q through Q+175.

2.5.9 PAPER TAPE VISUAL HEADER GENERATOR

The visual header generator is supplied in DBOS to permit labeling of paper tapes. The user may enter a series of characters which will be punched as a header in ticker tape visual format.

The header routine is given the name HDR in the DC file. Thus, it may be called to execution by the command:

$HDR

HDR use the following logical units:

OM - Operator error messages

PP - Punched output

15 - Compare input via PR

CC - Header data input

Generate Compare Control

If $15=NO, HDR will punch a header in visual format consisting of characters input from CC prior to the terminating C/R. If $15=PR, HDR will compare the tape in PR to the input data.

Error Detection

HDR will abort and return to DBOS if the input tape does not compare. The message ERROR is output to OM.

Character String Format

The following characters are valid:

A-Z 0-9 (). -/ and space.

Trailer may be included in the header by following the last trailer blank character with a # symbol. The special character @ will cause 8 rubout frames to be punched.

2.7 MEDIA DATA RECORD FORMATS

2.7.1 CARD DATA

"A" Format - 80 Hollerith characters.

"B" Format - Assembler and core image converter 54 word binary format (col 1 - 72) This format is the primary data storage unit for programs stored in the DC file. Each 54 word record contains an address and checksum. The checksum includes the relative sequence number of the record within the program file. The checksum is verified during all load operations.

"C" Format - This format consists of a packed format which results in 60 binary words per punched card. No address or checksum appears in the format.

"D" Format - Not applicable to cards.

2.7.2 PAPER TAPE FORMATS

"A" Format - Formatted ASCII data records. Each record must appear as follows:

LF, DATA, TRAILING BLANKS (which are omitted
by DBOS) CR

The LF is not packed as part of the data record and leader is slewed. A maximum of 80 characters can be input per record. A CR will terminate input and blank fill the remainder of input buffer.

"B" and "C" Format - In these formats the first frame contains a word count for the following record. Data is packed two frames per word. Leader is ignored.

Data transfer terminates when the DBOS word count, 54 or 60 words is reached. If the frame count is exhausted before the word count, the remainder of the input buffer is filled with zeroes.

2.7.3 DISK FORMATS

Data is stored on disk in four formats. It is generally, not acceptable to mix formats within files. The user should consider the format of data to be copied to/from a general purpose device. Refer to table 2-2 for implicit data type for each device.

A Format - Data stored in source form on disk are in packed ASCII format. Blanks are compressed and each input record requires a variable amount of disk storage. An average of 15 records are stored per sector.

B Format - The 54 word binary storage format is the primary unit under DBOS. A $COPY command or $DUMP command which references a DC or library file will result in 54 word records as output. (The user should note that at no time is a program stored on disk in core image format. Section 2.7.4 diagrams a method for obtaining a hexadecimal dump of a program in core image format.) All loading operations process the "B" format and each record is checksummed. Five records are stored per sector.

C Format - Sixty word binary record format. This format permits unchecksummed data to be processed.

Five records are stored per sector.

D Format - Sector data format. Each record contains 320 words. Files consists of multiples of disk sectors. A file may be defined by its name or its beginning, bs, and ending sector es.

2.8 ESTABLISHING NEW FILES

A new file may be defined at any time as a function of the JOB stream. For exampl e:

$SI=DS(800-1E82),[P]

would define a directoried source file ,DS, origined at sector 800 and terminating at sector 1E82. The P option defines the new DS definition as standard for multiple JOBs. A temporary assignment is reset on occurrence of the next JOB command. The user should also be aware that reloading the system, IPL, resets all LUN assignments to the basic set.

To add a source deck to the new file the command:

$COPY,CR,DS(NAME)

will add the label NAME to the directory for the file and store the source.

Any definition of a file in terms of beginning and ending sector establishes new temporary or standard limits for that file. A subsequent reference to DS in the example above is to the file bounded by 800 and 1E82 and not the basic set of boundaries (900-FFF).

To reset the boundaries of DS to the basic limits

```
$SI=DS (900-FFF)
```

The reference to SI is for formatting only. The file DS is the effected entity. SI may be reassigned without affecting DS.

NOTE

Care should be exercised when enlarging a file to not overlay another file. Considerable disk space is available outside the basic limits for user files. The user may reassign all file boundaries at any time.

NOTE

User library files must be terminated with an EOD image.

Example:

$JOB	
$2=UL(bs-es)	define limits for user library
$A	build library data records
Source Statements	
$A	
Source Statements	
$EOD	close file
$COPY,WB,UL	copy assembler generated subroutines and EOD to UL.

2.8.1 REASSIGNING FILES

In the previous section on defining new files the processes required to reassign files were defined. It is permissible to reassign files during a JOB stream repeatedly. This feature makes it possible to assemble or compile from several source files to build a particular object program. Example:

$SI=DS(NAME)	Unit 1 from basic DS file
$A	object to WB
$1=DS(bs-es)	Set SI to Fortran source file
$SI=DS(NAMEF)	Set SI to named segment of Fortran file
$F	Compile unit 2
$SI=DS ($bs_2$-$es_2$)	Set SI to source file 2
$A	Assemble unit 3
$EOD	Close binary object file
$CIC	Build program
*BUILD	
$LOAD	Execute program

Table 2-1. DBOS Logical Units

NUMBER	NAME	USAGE	STANDARD ASSIGNMENT (FILE) (SEE FILE NAMES, TABLE 2-2)
0	CC	Control Command Input	TY (Teletype Keyboard)
1	SI	Symbolic Input	CR (Card Reader)
2	SO	Symbolic Output	WS (Working Symbolic File)
3	BI	Binary Input	WB (Working Binary File)
4	BO	Binary Output	WB (Working Binary File)
5	LO	Listing Output	LP (Line Printer)
6	IS	Intermediate Symbolic	WS (Working Symbolic File)
7	OM	Operator Messages	TY (Teletype Printer)
8	CI	Core Image Data	WC (Working Core Image File)
9	LB	Binary Library	LB (Directoried Library File)
10	SL	System Log	LP (Line Printer)
12	SB	Secondary Binary Library	CR (Card Reader)
11	UL	User Library	UL (Directoried User Library File)
13		User Disk Temporary	DK (Unformatted Disk I/O File)
14		User Packed Disk Temporary	DP (Disk)
15		NO	NO (Delete I/O)

Table 2-2. DBOS File Names

NAME	DEVICE	USAGE
DS[1]	Disk	Directoried source language program data
LB[2]	Disk	Directoried binary object subroutine library
UL[2]	Disk	Directoried user binary object subroutine library.
DC[3]	Disk	Directoried core image program data
WS[1]	Disk	Working source language data
WB[2]	Disk	Working binary object data
WC[2]	Disk	Working core image data
CR	Card Reader	ASCII or binary card input
CP	Card punch	ASCII or binary card output
LP	Line Printer	ASCII listing output
TY	Teletype	ASCII or binary teletype input/output
PR	Paper Tape Reader	ASCII or binary high-speed paper tape input
PP	Paper Tape Punch	ASCII or binary high-speed paper tape output
DK[3]	Disk	ASCII or binary disk sector input/output
DP[2]	Disk	ASCII or binary logical packed disk input/output
NO	None	Delete input or output

1 ASCII character string 80 character records. Data type A in file manipulation commands.

2 Binary object format 54 word records. Data type B in file manipulation commands.

3 Binary data in 320 word record form.

2.9 JOB CONTROL FROM DIRECTORIED DISK FILE

2.9.1 DIRECTORIED JOB FILE

The directoried job file DJ is functionally identical to the DS file. Data is entered into the file by use of the copy command.

$$\$COPY \begin{bmatrix} ,A \\ ,B \\ ,C \\ ,D \end{bmatrix} ,file_1 \begin{bmatrix} (name) \\ bs\text{-}es \end{bmatrix} \begin{bmatrix} ,A \\ ,B \\ ,C \\ ,D \end{bmatrix} ,DJ(NAME)$$

The use of A format is mandatory for job string storage. (Note: The file may be used for general storage of any type data when not in use for directoried job string storage. The DJ file has an implicit mode of A.

Job strings stored in DJ are invoked by assigning logical unit CC to DJ.

$CC=DJ(NAME)

The last command in the job string must be a command which will transfer control back to the teletype or another file in DJ.

Example:

$CC=DJ(NAME)

$CC=TY or $CC=DJ(NAME2)

SECTION 3
DBOS OPERATIONS

3.1 DBOS OPERATIONS

DBOS monitor operation may be initialized by a bootstrap load from disk or, when the monitor is resident in core memory, by a manual interrupt or a programmed return to the monitor from an executing program.

3.1.1 BOOTSTRAP LOADING FROM DISK

With DBOS resident on disk, the user may load and initiate execution of the monitor by executing the following bootstrapping procedure:

1. Make the disk ready for initial program load (IPL).
2. Unlock CONSOLE ENABLE and WSPB switches.
3. Set the following switches ON (down). All other switches should be OFF (up).
   a. RUN/IDLE
   b. SPO
   c. REGISTER SELECT switch 8
   d. HALT
4. Operate the following switches in the following order:
   a. Press and release RESET
   b. Press and release ENTER
   c. Reset HALT (up)
   d. Press and release IPL
   e. Set HALT (down)
   f. Press and release RESET
   g. Reset REGISTER SELECT switch 8 (up)
   h. Press and release ENTER
   i. Set RUN/IDLE switch to RUN
   j. Press and release STEP

Following step 4j the bootstrap loader begins execution and loads the DBOS monitor and logical I/O system into core from disk and transfers control to the monitor.

When the monitor begins execution following this bootstrapping process, a series of initialization functions are performed:

1. Initialize channels.

   Zeroes are placed in channel address registers.

2. Set interrupt addresses.

   Pointers to interrupt handlers are placed in the interrupt addresses.

3. Unmask interrupts.

   All interrupts are made operational.

4. Execute executive.

   The system executive is loaded from disk and executed. (See executive operation, paragraph 3.2.)

All standard logical unit assignments are in effect following bootstrap loading of the monitor except SL which is assigned to TY.

3.1.2 CONSOLE INTERRUPT TO THE MONITOR

The operator may interrupt computer activity at any time and return control to the monitor by pressing the console interrupt switch. This causes the following:

1. The console interrupt handler transfers control to the monitor.
2. The monitor assigns the teletype keyboard as the CC device.
3. Memory is written to disk beginning at sector 75.
4. The message ***CONSOLE INTERRUPT is output to the teletype printer.
5. The monitor performs the four initialization functions listed in subparagraph 3.1.1. Only the CC logical unit assignment is reset.

3.1.3 PROGRAMMED RETURN TO MONITOR

Two methods are provided for returning control to the monitor from an executing program or processor:

1. Normal return. Normal return to the monitor is accomplished by a CALL to MON. A FORTRAN program should use the STOP statement which closes any logical files opened by the program before returning to the monitor.
2. Error return. Error return to the monitor is accomplished by a CALL to MONE. This return is the same as a normal return except an error message is output on OM.

Upon either return the monitor writes the entire memory to disk and performs the four initialization functions listed in 3.1.1.

3.1.4 PROGRAM RESTART THROUGH THE MONITOR

Monitor operations provide a method for restarting a user program or system processor. This is used primarily for program checkout purposes. This operation uses a program restart address established in the monitor at the time a console interrupt or error return situation occurs.

The symbolic location RSTRT (location 74) is the monitor entry for program restart.

To perform a program restart, the operator executes the following console operations:

1. Unlock the CONSOLE ENABLE switch.
2. Place the computer in the idle mode by setting the RUN-IDLE switch to IDLE.
3. Set the HALT switch to the lower position (ON).
4. Press and release the RESET switch.
5. Set the REGISTER SELECT switches to /0000 (all switches in the upper position) to select the I-register.
6. Set the console data switch to /0074 (RSTRT location of the monitor).
7. Press and release the ENTER switch.
8. Set the RUN-IDLE switch to RUN.
9. Press and release the STEP switch.

This procedure transfers control to the monitor at its RSTRT point. The monitor initializes interrupt conditions and restarts the user's program at the point at which operation previously ceased.

3.1.5 MANUAL ENTRY TO THE MONITOR

In addition to the manual restart of a user program or system processor and console interrupt , the operator may initiate monitor operation manually at either the normal or error operating level.

The following steps achieve manual entry:

1. Unlock the CONSOLE ENABLE switch.
2. Place the computer in the idle mode by setting the RUN-IDLE switch to IDLE.
3. Set the HALT switch to the lower position.
4. Press and release the RESET switch.
5. Set the REGISTER SELECT switches to /0000 (all switches in the upper position) to select the I-register.
6. Set the console data switch to /00XX, where 'XX' is one of the following:

   71 Normal console entry. Achieves the same operations as a normal return to the monitor.

   73 Abort console entry. Achieves the same operations as an error return or console interrupt to the monitor.

   74 Restart console entry. (See subparagraph 3.1.4.)
7. Press and release the ENTER switch.
8. Set the RUN-IDLE switch to RUN.
9. Press and release the STEP switch.

This procedure transfers control to the appropriate monitor entry point and initiates monitor execution. Except for RSTRT (location 74) entry to the monitor transfers control to the executive (see executive operations, paragraph 3.2).

3.1.6 MONITOR FIXED LOCATIONS

The monitor contains a series of fixed locations utilized as entry points (previous paragraphs) or as storage locations for pertinent program information. These locations are shown in table 3-1.

Table 3-1. Storage Locations for Pertinent Program Information

HEXADECIMAL LOCATION	NAME	USAGE
70	MON	Normal program return to the monitor to transfer control to the executive. CALL MON
71		Normal console entry to the monitor to transfer control to the executive.
72	MONE	Abort program (error) entry to the monitor to transfer control to the executive. CALL MONE
73		Abort console entry to the monitor to transfer control to the executive.
74	RSTRT	Console entry to the monitor to initialize interrupt conditions and restart the user's program or system processor in core.
77		Location register saved. In cases of program interruption or error abort, this location contains the address of the next instruction following the interrupted instruction.
78		A register save location.
79		Q register save location.
7A		Index 1 save location.
7B		Index 2 save location.
7C		Index 3 save location.
85	LIO	Entry point to LIO.

3.2 EXECUTIVE OPERATIONS

Following entry into the monitor (except for program or processor restart), the monitor loads and transfers control to the executive. The executive initializes the I/O channels and the busy indicators of the I/O drivers and then begins processing control commands. Control commands are read from logical unit CC.

When the teletype keyboard is the CC device, the executive requests control command input by output of a line feed, return, a ? and space on the teletype printer and waits for a key-in. (See subparagraph 2.2.1, control commands input from the teletypewriter.)

The teletype keyboard is the standard CC assignment (see table 2-1) and retains this assignment unless changed by a '$lun=file' control command (subparagraph 2.4.1). If the CC device assignment is changed, the assignment reverts back to the teletype keyboard under one of the following conditions.

1. $CC=TY

   Control command unit reassignment.

2. Console interrupt.

3. Any one of the processing errors or interrupt conditions listed in subparagraph 3.3.3.

The executive processes control commands as described in section 2. It does not remain resident in core during execution of a user program or a system processor. It is reloaded by the monitor as needed to process control commands.

3.3 SYSTEM MESSAGES

DBOS provides output messages to indicate device not ready and error or interrupt circumstances. These messages fall into three categories:

1. Input/output messages.
2. Control command error messages.
3. Processing error and interrupt messages.

3.3.1 INPUT/OUTPUT ERROR MESSAGES

These messages are printed on the console teletype printer and reflect conditions of devices which require operator action.

MESSAGE:

file NOT READY

'file' is a file name as defined in table 2-2. This message indicates that a device is not turned on or is not ready due to a lack of physical data (e.g., card hopper empty).

When the operator has made the device READY, operation resumes automatically.

MESSAGE:

file ERROR

'file' is a file name as defined in table 2-2.

This message indicates a problem condition with the device.

Following output of this message, the system requests a key-in by the operator to indicate the next action. The following alternatives may be taken:

ACTION	KEY-IN
Retry the operation	R return
Ignore the error	
Ignore the error, proceed (continue)	C return
Abort the process, return to the executive	Any other character

3.3.2 CONTROL COMMAND ERROR MESSAGES

These messages are listed on the teletype printer and also on the SL device (if SL is not assigned to teletype). Upon printing these messages, the system continues to read the next control command.

**SYNTAX	The statement is syntactically incorrect
**FILE(NAME) DEFINED	The output file of a COPY command references an existing entry in a directoried file. Change name of program to be copied or use REPLACE command.
**UNDEFINED FILE	An undefined file name was specified.
**UNDEFINED NAME	The program (processor) specified in a $NAME command is not in the DC file directory. COPY program to DC file or use $LOAD if program is in WC file.
**INVALID ADDRESS	The 'bs-es' sector address or the 'loc_1-loc_2' memory address specified is incorrect.
**INVALID CC	A CONTROL command was not preceded by a $.
**INVALID LUN	An illegal logical name or number was specified, refer to table 2-1.

**ILLEGAL OPERATION	Operation is valid only for directoried files. Correct file assignment or procedure.
**PARAMETERS IGNORED	Notification only. A 'bs-es' was specified for a non-disk file.
**INVALID DELIMITER	A character other than blank, comma, dash or parenthesis was contained in the control command. Correct control record or input correctly.
**UNUSABLE DISK PACK	More than five defective sectors were encountered during disk initialization. Retry initialization and/or replace disk pack.
**DEVICE FAIL	Disk error. Attempt operation again.
**DISK UNIT UNDEFINED	Either the type of disk drive has not been defined or the disk has not been initialized. Refer to section 2.5.7 for disk initialization.
**CHECKSUM ERROR	A checksum error has occurred while copying the Executive to disk. Check sequence number on cards for correct order and retry.

**CORE SIZE UNDEFINED	An attempt has been made to write the Monitor to disk without defining the size of available core. Refer to section 2.5.7, $CORE.
**MONITOR NOT ON DISK	An attempt was made to cole start a new system with $START command prior to writing Monitor to disk. Refer to section 2.5.7, $WMON.
**EXECUTIVE NOT ON DISK	An attempt was made to cold start a new system with $START command prior to writing the Executive to disk. Refer to section 2.5.7, $EXEC.

3.3.3 PROCESSING ERRORS

Processing errors indicate a failure during processing of user programs. When these occur the operation is terminated and control is returned to the executive with CC forced to the teletype keyboard. Since memory is written to disk when such a return occurs, a memory dump can be obtained using the PDUMP control command.

Processing error messages are listed on the teletype printer and on the SL device (unless SL is assigned to teletype).

**PROCESSING ERROR. Control was transferred to the monitor abort location.

**LOAD ERROR. An error occurred during loading of a system processor or a user program.

**CONSOLE INTERRUPT. The console interrupt switch was pressed. (See note below).

**ILLEGAL INSTRUCTION EXECUTION. An operation code not included in the GA 18/30 instruction set was encountered. (See note below.)

**POWER FAIL. Power to the hardware system falls below minimum limit. (See note below.)

**RESTART. Processing is automatically restarted following power failure. (See note below.)

**MEM PARITY. Memory parity check indicates incorrect read-out of a memory word. (See note below.)

**MEM PROTECT VIOLATION. Attempt made to invade protected memory. (See note below.)

**DATA CHANNEL ERROR. Hardware error in data transmission. (See note below.)

NOTE

In cases where an error condition or operator intervention causes an internal interrupt, the message

**AT LOCATION XXXX

is printed. XXXX is the hexadecimal address of the next instruction following the instruction at which interrupt occurred.

SECTION 4
SYSTEM GENERATION

4.1 GENERAL

The GA DBOS is supplied to the user as a deck of cards or several rolls of punched paper tape. All standard components for generating the system are supplied.

The following procedures are designed to make the task of system configuration and generation as minimal as possible. The SYSGN program consists of a set of utilities which are used initially to build a DBOS, but which may also be used for later modification of the system. The following paragraphs detail the steps which must be executed and in what order. Those steps which are optional are bracketed and may be executed in any order. Refer to section 2 for details of SYSGN utilities. For paper tape procedures skip to section 4.7.

4.1.1 CARD SYSTEMS

The following components are included in the supplied card deck and appear in the following order:

1. Bootstrap program
2. Monitor, logical I/O and system generator
3. Executive
4. Standard processors (order is arbitrary)
    a. Assembler
    b. FORTRAN compiler

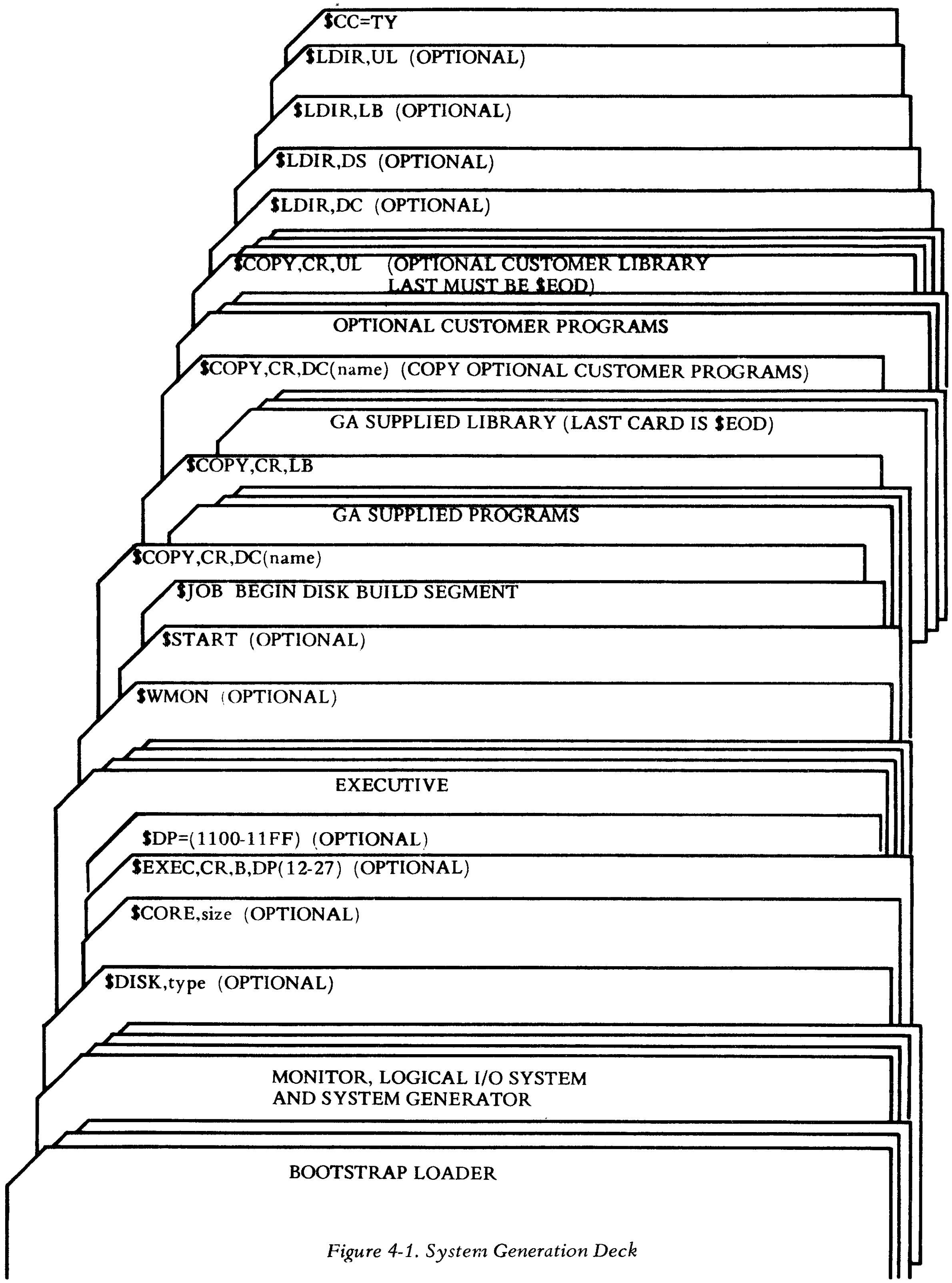


Figure 4-1. System Generation Deck

c. Core image converter

d. Editor

e. Debug routine

f. System generation utility

g. Sequence/compare utility

h. Bootstrap generator

i. Header generator

5. System library

Each processor is in absolute format, directly storeable into directoried files. The following paragraphs describe the procedures for generating the disk system from the supplied deck.

4.2 CONSOLE BOOTSTRAP PROCEDURE

To initiate system generation, the operator execute the following steps:

1. Load the system on the card reader and make the reader ready.
2. Unlock CONSOLE ENABLE and WSPB switches.
3. Set the following switches ON (down). All other switches should be OFF (up).
   a. RUN/IDLE
   b. SPO
   c. REGISTER SELECT switches 8 and 4.
   d. HALT
4. Operate the following switches in the following order:
   a. Press and release RESET
   b. Press and release ENTER
   c. Reset HALT (up)
   d. Press and release IPL
   e. Set HALT (down)
   f. Press and release RESET
   g. Reset REGISTER SELECT switches 8 and 4
   h. Press and release ENTER
   i. Set RUN/IDLE switch to RUN
   j. Press and release STEP.

NOTE

At this point control is transferred to the bootstrap program.

4.3 BOOTSTRAP PROGRAM EXECUTION

Execution of the system generation bootstrap program loads the system generator/monitor/logical I/O segments of the system from the card reader into core. When this loading is complete, control is transferred to the system generator.

If the computer halts during this loading process, a checksum error has occurred. No message is printed. To recover from this situation, the system generation must be begun again with the console bootstrap procedure (paragraph 4.2).

4.4 SYSTEM GENERATION EXECUTION

When the bootstrap program transfers control to the system generator, the following message is printed on the teletype printer:

BEGIN SYSTEM GENERATION

The system generator then waits for an operator response.

Four basic steps are involved to accomplish system generation.

1. Define hardware and executive storage.
2. Define DBOS characteristics.
3. Storage of Monitor.
4. Loading of processors.

4.4.1 HARDWARE DEFINITION

Enter the following commands through the teletype keyboard. (See section 4.6 for card control method.)

$DISK,unit	1341, 1344 or 1345
$CORE,size	8K, 16K or 32K
$IDISK	Initialize entire disk

4.4.2 STORE EXECUTIVE

Enter the following command:

$EXEC,CR,B,DP(12-27)

When loading is complete control will be returned to the teletype. Enter the following command:

$DP=(1100-11FF)

4.4.3 DBOS CHARACTERISTIC DEFINITION

The monitor supplied with the system is complete and ready to operate. The logical unit assignments in effect after a $JOB command is processed are detailed in Appendix F. The assignment of logical unit SL is to the teletype after an Initial Program Load operation and prior to execution of a $JOB. The standard disk file assignments are outlined in appendix A of this manual. In some cases, however, the user may require a different file structure of logical unit definition. For example when the line printer operations are included, but no line printer is present, these operations must be removed by file redefinition. The following commands are all optional. If none of the options are desired, skip to section 4.4.4.

$file(bs-es)	change disk file limits
$lun=file[bs-es],P	change initial and/or standard logical unit assignments

4.4.4 STORE MONITOR AND EXECUTE

The fully configured monitor is written to disk by use of the following command:

$WMON

This command writes the system bootstrap and monitor to disk.

Initial System Operation

$START

This command transfer control to DBOS. All modifications made with any of the SYSGN commands will be in effect immediately.

4.5 COMPLETION OF THE SYSTEM GENERATION

The generation of additional elements (processors) into DBOS is a function of the executive. Therefore, after the executive has been read into core and successfully written on disk, execution of the executive begins.

Control commands (section 2, paragraph 2.4) to the executive are read from the system generation input device (card reader) preceding each processor. The appropriate commands are supplied with the system deck. As many processors as desired may be generated into the system at this time if each is preceded by a properly configured COPY command.

Enter one of the following commands to initiate loading of processors:

$CC=CR for card systems

$CC=PR or TY for paper tape systems

The user may store programs in the directoried object program file, DC, and create a secondary library, UL. Refer to section 2.5.3 for details of UL usage.

To create a user library file a copy command of the form $COPY,CR,UL must precede the actual library in the card stream. See section 2.4.3 for use of copy command.

To create a DS file the user must precede each program source deck to be included in DS with a copy command.

The copy command is, $COPY,CR,DS(NAMEX).

When a user supplied file is stored during generation, a list directory command should be included to map the file. Refer to figure 4-1.

The last control command read from the system generation deck should be:

    $CC=TY

As the elements are read into core and written on disk, the executive builds a directory on disk containing the element names and the disk storage area (beginning and ending sectors) of each.

Following the terminating library $EOD control command, a series of $LDIR commands causes the system directories to be printed on the SL device (line printer or teletype printer). (See list directory command ($LDIR), section 2, subparagraph 2.4.6.)

When the $CC=TY command is encountered, this command is printed on the SL and OM devices (line printer and teletype printer, if both devices are available) and then requests control command input from the CC device (teletype keyboard; the $CC=TY reassigns CC to the teletype by output of a Line Feed, Return, ? and Space on the teletype printer).

The executive then waits for operator response. At this point, system generation is complete and operation, using DBOS, may begin either by issuing control commands to the executive or by executing a bootstrap loading procedure from disk (section 3, paragraph 3.1).

NOTE

It will be recognized by the user that during generation of DBOS, the system generator is used only to generate the monitor, logical I/O, and executive portions of the system and to initiate execution of the executive for completion of system generation. Therefore, the makeup of the system beyond these basic elements is optional. However, the user should be thoroughly familiar with the executive control commands before altering or adding to the standard, delivered system.

4.6 CARD CONTROLLED SYSTEM GENERATION

The DBOS deck as supplied returns control to the teletype keyboard after the monitor, LIO and system generator have been input so that user supplied instructions may be input.

Total batch generation via the card reader is possible if the following steps are taken:

1. Insert the following cards in front of the cards remaining in the reader:

   $DISK,unit

   $CORE,size

   $IDISK
   $EXEC,CR,B,DP(12-27)
   $DP=(1100-11FF)

2. Locate the last card of the executive deck and insert immediately after the following optional cards if any:

   $file(bs-es)

   $lun=file [bs-es] ,P

   $IDIR,file

   $D

3. Insert the following cards after those in section 2.

   $WMON

   $START

4. Enter the following command via the teletype keyboard:

   $CC=CR

5. Control will return to the teletype when generation is complete.

4.7 PAPER TAPE SYSTEMS

The GA 18/30 DBOS is supplied to the user as a set of paper tapes. All standard components for generating the system on disk are included and are grouped as follows by individual tape:

Drawing No./Label	Description
94Z00138A02	Bootstrap program, Monitor, logical I/O and system generator
94Z00138A03	Executive
94Z00138A04	Editor, assembler, debug routine, core image converter and system generator utility
94Z00138A05	Header utility, boot utility and sequence/compare utility
94Z00138A06	FORTRAN Compiler - parts 1 and 2
94Z00138A07	System library
	SPC-12 Cross Assembler
	SPC-16 Cross Assembler and Simulator

If loading is to be accomplished via the TTY reader, refer to the 18/30 Programming Operations Manual, 88A00121A, Appendix G for bootstrap procedures.

4.7.1 CONSOLE BOOTSTRAP PROCEDURE

To initiate system generation, the operator must execute the following steps:

1. Load the tape labeled 94Z00138A02 into the paper tape reader and make the reader ready.
2. Unlock CONSOLE ENABLE and WSPB switches.
3. Set the following switches ON (down). All other switches should be OFF (up).
   a. RUN/IDLE
   b. SPO
   c. REGISTER SELECT switches 8 and 4
   d. HALT
4. Operate the following switches in the following order:
   a. Press and release RESET
   b. Set data switch 3, 4, 5, 6 and 7 down (1F00)
   c. Press and release ENTER
   d. Reset HALT (up)
   e. Press and release IPL (indicators = 1078 when finished)
   f. Set HALT (down)
   g. Make disk ready
   h. Press and release RESET
   i. Reset REGISTER SELECT switches 8 and 4
   j. Press and release ENTER

k. Set RUN/IDLE switch to RUN

l. Press and release STEP

Note: At this point, control is transferred to the bootstrap program.

4.8 BOOTSTRAP PROGRAM EXECUTION

Execution of the bootstrap program loads the System Generator/Monitor/Logical I/O segments of the system from the paper tape reader into core.

When the program has been loaded into core, a WAIT at location /103F will occur. At this point, set REGISTER SELECT switch 4 down. If the register indicators display all zeros, the load was successful - any other value indicates a checksum error which requires restarting with the console bootstrap procedure.

If a successful load is indicated, set REGISTER SELECT switch 4 up and data switch 0 down. Press STEP to transfer control to the system generator.

4.9 SYSTEM GENERATOR EXECUTION

When the bootstrap program transfers control to the system generator, the following message is printed on the teletype printer:

```
BEGIN SYSTEM GENERATION
?
```

The following commands must be input via the teletype keyboard:

$DISK,unit	1341, 1344 or 1345
$CORE,size	8K, 16K or 32K
$IDISK	Initialize entire disk

4.10 LOADING THE EXECUTIVE TAPE

Load the tape labeled 94Z00138A03, Executive into the tape reader with the blank leader following the title under the read head.

Enter the following command via the teletype keyboard:

$EXEC,PR,B,DP(12-27)

1. The Executive is then read from the paper tape reader and written to disk.
2. If a checksum error is encountered while reading this or any other processor, the message:

   **CHECKSUM ERROR

   is printed on the teletype printer. The only alternative is to restart the loading process for that particular tape.
3. If the Executive was successfully written to disk, the teletype printer will then print a question mark and wait for operator response.

Enter the following command:

$DP=(1100-11FF)

4.11 DBOS CHARACTERISTICS DEFINITION

The monitor supplied with the system is complete and ready to operate. The logical unit assignments in effect after a $JOB command is processed are detailed in table 2-2. The assignment of logical unit SL is to the teletype after an IPL operation and prior to execution of a $JOB. The standard disk file assignments are outlined in appendix A of this manual. In some cases, however, the user may require a different file structure or logical unit definition. (For example that the line printer is included and must be removed from operation by file redefinition if not present on the object system.) The following commands are optional, if none of the options are required skip to section 4.12.

$file(bs-es)	change disk file limits
$lun=file [bs-es] ,P	change logical unit assignments

4.12 STORE MONITOR AND EXECUTE

The fully configured monitor and bootstrap loader are written to disk by use of the following command:

```
$WMON
```

Initiate System Operation

The following command places the defined monitor into execution:

```
$START
```

4.13 COMPLETION OF SYSTEM GENERATION

Section 4.5 describes the general procedures for completing system generation. The procedures for paper tape generation differ on slightly from those used for cards.

A separate control command must be entered via the teletype keyboard to load each processor object tape:

    $CC=PR

Tapes supplied in more than one part must be processed in order (FORTRAN compiler - parts 1 and 2). The system library must be processed last.

Refer to section 4.5.

4.14 DBOS CONFIGURATION KIT

The DBOS Configuration Kit provides the means for altering DBOS to suit an installation's particular input/output structure. The following capabilities are included:

1. Add or remove standard I/O device drivers.
2. Add user written device drivers for any I/O device.
3. Include a second system disk buffer for increased disk transfer rates.
4. Add disk files.

The Configuration Kit consists of the following on cards or paper tape:

1. The System Monitor (relocatable object).
2. A library of standard I/O device drivers and monitor components (relocatable object).
3. A library containing SYSGN components and the DEBUG subroutines (relocatable object).
4. The configuration subroutine CONFIG(source). CONFIG is a portion of the Resident Monitor and contains:
   a. Logical Unit table
   b. Physical Unit table
   c. System disk packing buffers.

4.14.1 LIO OPERATIONS

All input/output under DBOS is performed on logical units. A logical unit is known only by its logical unit number or logical function name. Logical function names are permanently equated to a corresponding logical unit number as listed below:

CC =	logical unit 00
S1 =	logical unit 01
S0 =	logical unit 02
BI =	logical unit 03
B0 =	logical unit 04
L0 =	logical unit 05
IS =	logical unit 06
OM =	logical unit 07
CI =	logical unit 08
LB =	logical unit 09
SL =	logical unit 10
UL =	logical unit 11
SB =	logical unit 12

There are three additional logical units included in the standard system. These may only be referred to by their numbers (13, 14, and 15).

When LIO processes an I/O operation it connects a logical unit number or name to a device driver in the following manner:

1. Given a logical unit number (00-15) or logical unit name (CC,SI,etc.) refer to the Logical Unit table for a pointer associating that logical name/number with an entry in the Physical Unit table.
2. The entry in the Physical Unit table will contain the address of an I/O List which defines the operation.
3. The I/O List contains the address of a logical unit driver (a program which performs the input/output function).
4. Control is given to the logical driver for the operation. Communication between LIO and the logical driver is maintained via the I/O list.
5. The logical driver may in turn call another program, the Physical device driver, which actually operates the device. The logical and physical drivers also communicate via the I/O list.

The physical driver is the lowest level program in the chain. It may be used by many logical drivers. A logical driver may be used by many I/O lists. An I/O list may be used by many logical unit names/numbers.

The functions of the physical driver may be incorporated into the logical driver, thus eliminating the need for a separate physical driver.

The separation of logical and physical driver simplifies configuration of non-standard devices. Usually, only a new physical driver need be written functionally replacing an existing physical driver. However, this approach requires more core than writing a single I/O driver. The I/O list contains parameters for both logical and physical drivers. Some parameters may be omitted if the I/O driver is coded as a single entity.

4.14.2 LOGICAL UNIT TABLE

The Logical Unit Table contains ordered logical unit assignments and appears as:

LNMAX	DC	LUTBE-LUTB	Number of entries
LUTB	DC	/ppcc	Physical Unit Table assignment for logical unit 0
	DC	/ppcc	Physical Unit Table assignment for logical unit 1
	.		
	.		
	.		
	DC		Physical unit table assignment for last logical unit
LUTBE	EQU	*	

The pp and cc are, respectively, the permanent and current assignments of logical units to Physical Unit Table entries. There is not a one to one correspondence between logical entries and physical entries. The standard system contains 16 logical unit entries. If more are desired, it is necessary only to add additional entries following the standard entries in the form:

```
        DC     /ppcc
```

where:

pp = a value equal to twice the physical unit table entry number to which this logical unit is to be permanently assigned (i.e., if the physical entry number is 18 this value is 36)

cc = a value equal to twice the physical unit table entry number to which this logical unit is to be temporarily assigned (usually identical at generation time to the permanent assignment).

If the Physical Unit Table has been altered from standard by deletions, insertions or replacement it may be necessary to change the standard logical unit assignments. For example, logical unit CC is normally assigned to the tenth entry, the TTY, in the physical unit table. If, due to re-sequencing, the tenth entry is a non-input device, the system would fail.

4.14.3 PHYSICAL UNIT TABLE

The Physical Unit Table (within CONFIG) must have an entry for each file name and its associated device list. Each entry consists of two words:

1. The file name - this must be two letters preceded and followed by periods.
2. The name of the I/O List which defines the operation.

The Physical Unit Table appears as:

```
PUMAX    DC    PUNME-PUNM
PUNM     ASC   .$$.         System Dummy
         DC    0
         ASC   .name1.      File name1
         DC    list1        List name1
         .
         .
         .
         ASC   .namen.      Last file name
         DC    listn        Last list name
PUNME    EQU   *
```

The usual case is to have the lists external to CONFIG (the lists are kept in file UL). Thus, for each external list name a REF must be included in CONFIG, e.g., REF $\quad name_1$

REF $\quad name_2$

REF $\quad name_n$

It is advisable to not remove or replace entries in the table. New entries should not be inserted between existing entries but rather at the end of the table. This prevents the necessity for altering the Logical Unit Table for standard assignments.

If, however, it is desired to alter the sequence of the Physical Unit Table by replacement, removal, or insertion, the following considerations apply:

1. The system dummy must be maintained as entry zero.
2. The third entry must be .DC.
3. The sixth entry must be .WC.
4. The deflections in the Logical Unit Table must be coordinated with the new Physical Unit Table sequence. (See Logical Unit Table.)

4.14.4 I/O LISTS

An I/O List is a block of data which acts as the link between an I/O request, the logical driver, and the physical driver performing an I/O operation.

When adding a new driver an I/O List must be inserted into the library to correspond with an entry in the Physical Unit Table. Note that appropriate REFs and DEFs must appear in the Physical Unit Table, I/O Lists, and drivers to effect linkage.

See figure 4-2 for an outline of an I/O List. The parameters are explained below:

M0 Physical List Busy

Set/Reset by physical driver

0 - not busy

$\neq$ 0 - busy

M1 Physical Driver OPCOP

The OPCOP parameter may be used to specify the address of a subroutine to be executed by the physical driver when the operation requested of the physical driver is complete. If the parameter contains a zero, no OPCOP subroutine entry is specified.

If the value of the parameter is non-zero, the physical driver assumes the value to be the entry address to a subroutine. The parameter sent to the OPCOP subroutine is the address of a word which contains the address of the I/O List which activated the physical driver

Physical Driver Parameters		DEF	name		
	name	EQU	*		I/O List name
		DC	0	(M0)	Physical Driver List Busy
		DC	popco	(M1)	Physical Driver OPCOP
		DC	pbusy	(M2)	Busy Location in Physical Driver
		BSS	3	(M3-5)	Unassigned
		DC	*-*	(M6)	Physical Driver Error Code
		DC	0	(M7)	Control Parameter
		DC	0	(M8)	Area Address
Logical Driver Parameters		DC	lopco	(L1)	Logical Driver OPCOP
		DC	driv	(L2)	Logical Driver Address
		DC	*-*	(L3)	Logical Driver Error Code
		DC		(L4)	Logical Device Characteristics
		DC	0	(L5)	Logical Function Code
Optional Additional Parameters		DC		(E1)	Available (used mainly for disk files)
		.			
		.			
		.			
		DC		(E_n)	

Figure 4-2 I/O List

operation. Entry to the OPCOP routine is made with a calling sequence as shown:

```
      BSI   I     M1
      DC          PARA
            .
            .
            .
      PARA  DC    LIST
```

The physical driver may again be called from within the OPCOP routine, only if all of the following are true:

1. The error parameter in the I/O List, M6, is set to a one (1), (operation complete).
2. The control parameter specified a valid op-code on the previous request, and
3. No request was made to the physical driver between the time of the initial call and the OPCOP exit.

Also, if a subsequent call is made to physical driver from within the OPCOP routine, no status waits or delays can be used since the OPCOP routine is entered in an interrupted state.

M2 Busy Location in Physical Driver

This entry in the I/O List contains the address of a location within the physical driver. The value in this location is set by the physical driver to zero (0) for not busy, non-zero for busy.

(M2) = 0 - busy

(M2) $\neq$ 0 - not busy

(Note that M0 and (M2) have reverse notations)

M3-M5 May be used for physical driver/logical driver communication.

M6 Error Parameter

When the physical list busy (M0) indicator is set to zero, following an I/O operation, M6 is set to one of the following values by the physical driver

1. Successful completion of the call.
2. Device logically disconnected from the system.
3. Device hardware non-ready.

>3. Any hardware malfunction which did not allow completion of the call.

Error values of greater than 3 generally have different meanings for each driver.

Note that it is the user's responsibility to initiate any desired error recovery procedures.

M7 Control Parameter

This parameter consists of four hexadecimal digits which define the I/O operation to the physical driver. The first two digits defines the I/O function in terms of read, write, control, etc. The second two digits modify the function digits.

If no physical driver exists, parameters Mo-M7 should be defined as follows:

```
M0        DC     0
M1        DC     0
M2        DC     *+1
M3        DC     1
M4        DC     *-*
M5        DC     *-*
M6        DC     0
M7        DC     0
M8        I/O area address of the form:
          AREA   DC    n      word count
                 BSS   n      data area
```

L1 Address of user OPCOP or zero. This contains the address of a subroutine in the user's program which is to be executed by the logical drive prior to returning to the user's program via LIO.

L2 Address of logical Driver

L3 Logical Driver error code

Must be set by logical driver according to table 5-2 in the DBOS Reference Manual.

L4 Device Characteristics

A word which contains the sum of one or more of the following characteristics

DR	EQU	/1	Directoried
IN	EQU	/4	Input device
OT	EQU	/8	Output device
BI	EQU	/410	Implicit binary mode
AL	EQU	/420	Implicit alpha mode
DK	EQU	/42	Disk
PK	EQU	/80	Packed format
LW	EQU	/100	Line width records
LB	EQU	/200	Library file

L5 LIO Operation code with logical unit number removed /$X_1X_2$00. (See table 5-1 in the DBOS Reference Manual).

4.14.5 ADDITION OF I/O DRIVERS

a. The Physical Unit Driver

A new physical unit driver which may be used by many logical drivers must include the following functions.

1. It's name must appear in a DEF as;

```
        DEF      PDNAM
```

2. It must reserve a location as a driver busy indicator. The address of this location must be made available to an I/OList via a DEF as:

```
        DEF      PDBSY
PDBUSY  DC       non-zero
```

where PDBUSY is maintained by the physical driver as zero for busy, non-zero for not busy.

3. It must accept a call from the logical driver of the form:

```
        BSI      PDNAM
        DC       LIST
```

The parameter being the address of M0 in the I/O List.

4. It must maintain M0 and M6 in the I/O List.

5. It must execute optional OPCOP if M1 in I/O List so specifies.

A physical unit driver which performs the I/O operation and the above actions is coded and placed in the User's Library. (See Section 4.14.10)

b. The Logical Unit Driver

A new logical unit driver which may be called by many I/O Lists must perform the following functions:

1. It's name must appear in a DEF as:

```
DEF          LDNAM
```

2. It must accept a call from LIO via an I/O List of the form:

```
BSI   I      L2
DC           LIST
```

where LIO sets the contents of I/O List locations:

M8 = AREA

L1 = OPCOP

L5 = FUNC

3. It must maintain L3 in the I/O List.

4. It must provide the physical driver with its control parameter (M7) using LIO supplied codes in L5.

5. It must call the physical driver, passing on the address of the I/O List

6. It must execute optional OPCOP if specified by LIO.

A logical I/O driver which performs the above functions as well as it's own operation (data conversion, etc.), is coded and placed in the User's Library.

c. The Combined Logical/Physical Driver

Much of the bookkeeping necessary for separate drivers is eliminated. A combined driver need perform only the following:

1. The actual I/O operation.
2. Numbers b-1, b-2, b-3 and b-6 above.

4.14.6 DELETION OF I/O DRIVERS

It is to the user's advantage to remove all standard drivers which do not pertain to his actual configuration. This will increase the amount of core storage available to user programs.

The following steps must be taken to remove unwanted drivers:

1. Remove from CONFIG source deck all references and entries pertaining to the drivers.

   a. Physical Unit Table - Remove the two word entry and the REF for that entry. Replace the two word entry with:

      ```
      ASC     .NO.
      DC      NOFL
      ```

      This null entry eliminates the need to alter deflections in the Logical Unit table.

   b. Logical Unit table - If any of the logical units are assigned to the deleted entry, this assignment should be changed to another physical unit.

2. Build a new system as shown in Section 4.14.10.

When the new system is built by CIC the unwanted I/O Lists and drivers will not be included.

The following example illustrates the removal of the line printer driver from the standard system:

1. In the Physical Unit table replace the ninth entry,

```
ASC    .LP.
DC     LPRT
```

with

```
ASC    .NO.
DC     NOFL
```

Also remove the entry:

```
REF    LPRT
```

2. Since logical units 05 (LO) and 10 (SL) are standardly assigned to the Line Printer, these entries in the Logical Unit table must be re-assigned to the TTY. This is accomplished by replacing respectively the entries:

```
DC     /1212     LO
DC     /1214     SL
```

with

```
DC     /1414     LO
DC     /1414     SL
```

3. Build new system as shown in Section 4.14.10.

Since no reference is made to LPRT the I/O List will not be included. Since the I/O List is not included there will be no reference to the driver and it also will not be included.

4.14.7 SYSTEM DISK BUFFERS

Standard systems are configured with one system disk buffer in the system monitor. An additional buffer may be added to reduce the time for disk to disk transfers. This requires only a change in CONFIG. To implement the second buffer replace the

```
SBUF2   EQU   SBUF1
```

card in CONFIG with:

```
SBUF2   DC    0
        DC    320
        BSS   321
```

Note that this will add 323 words to the resident size of the monitor - see Section 4.14.9.

4.14.8 ADDITIONAL DISK FILES

Additional drivers are not required to add more disk files. This can be accomplished by adding more I/O lists and entries in the Physical Unit Table. Disk files require additional I/O list parameters (E1-E10):

E1	-operation code of last operation
E2	-first sector
E3	-"now" sector
E4	-last sector
E5	-buffer
E6	-buffer pointer
E7	-user area save
E8	-first permanent sector
E9	-last permanent sector
E10	-system buffer
E11	-first directoried sector (directoried files only)
E12	-last directory sector (directoried files only)

Example:

To add a directoried file, DF, to be assigned to sectors 1C00-1CFF and use SBUF2:

1. Add file name and list name to the Physical Unit Table:

```
REF   DFFL
ASC   .DF.          File name
DC    DFFL          List name
```

2. Add I/O list:

```
           DEF    DFFL
           REF    LDZ
           REF    BSYDK
           REF    SBUF2
BS         EQU    /1C00
ES         EQU    /1CFF
DFFL       DC     0          M0
           DC     0          M1
           DC     BSYDK      M2
           BSS    3          M3-M5
           DC     0          M6
           BSS    2          M7-M8
           DC     0          L1
           DC     LDZ        L2
           DC     *-*        L3
           DC     /4EF       L4=DR+IN+OT+AL+DK+PK
           DC     *=*        L5
           DC     0          E1*
           DC     BS         E2*
           DC     BS         E3*
           DC     ES         E4*
           DC     SBUF2      E5*
           DC     SBUF2+3    E6*
           DC     0          E7
           DC     BS         E8
           DC     ES         E9
           DC     SBUF2      E10
           DC     BS         E11*
           DC     ES         E12*
           END
```

These parameters may be left undefined (-*) if a $JOB command is used before using the file. For safety, they should be defined as shown in the example.

4.14.9 OTHER CONSIDERATIONS

Adding more drivers or a system buffer increases the resident size of monitor. If this size exceeds the origin of an existing processor (or the executive), that processor must be re-origined (using the CIC BOUND command) above the monitor. This is done as follows:

1. Build new processor in WC
```
$JOB
$BI=CR
$CIC
*MAP
*BOUND, mon-end+1
*BUILD
PROCESSOR
$EOD
```

2. Punch new processor

   a. CARDS
```
$JOB
$SI=WC
$SO=CP
$SQCM
>name001,B
```

   b. PAPER TAPE
```
$JOB
$COPY,WC,PP
```

4.14.10 OPERATING PROCEDURES

Once the subroutine CONFIG has been modified and optional non-standard drivers have been written, the following must be done:

1. Add the GA configuration components to an existing DBOS pack:

$COPY,CR,UL	
GA supplied monitor components and drivers	Copy components to UL File
$EOD	Close UL File
$COPY,CR,LB	
SYSGN components and DEBUG sub-routines	Copy to LB File
$EOD	Close LB File

2. Optionally add user generated drivers and/or I/O lists to UL:

$JOB	Assemble drivers and I/O lists
$A	
User Component 1	
$A	
User Component 2	
.	
.	
.	
$A	
User Component N	
$EOD	Close WB File
$COPY,WB,UL	Copy drivers and lists to UL File

3. Build a new DBOS system in WC:

$JOB	
$A	
CONFIG	Assemble updated CONFIG
$EOD	Close WB File
$IS=WS	Assign CIC intermediate storage to WS File
$BI=CR	Assign BI to card reader
$SB=WB	Assign SB to WB (object CONFIG)
$CIC	
*MAP	
*BOUND,6C	
*BUILD,SB,UL	Build from BI, SB, UL and LB into WC
SYSTEM MONITOR	Monitor relocatable binary cards

4. Punch new system with bootstrap:

a. CARDS
   $JOB
   $SI=WC
   $SO=CP
   $BOOT
   $SQCM
   >MON001,B

b. PAPER TAPE
   $JOB
   $SO=PP
   $BOOT
   $COPY,WC,PP

SECTION 5
LOGICAL INPUT/OUTPUT SYSTEM

5.1 GENERAL INFORMATION

The logical input/output system (LIO) provides system capability for performing device independent input/output operations.

LIO data transmission is processed on a record basis and internal data is represented as ASCII information packed two characters per word or as binary word data. Conversion for particular devices is performed automatically.

LIO consists of three separate levels of processors:

a. The central control routine which processes user calls and converts logical unit numbers to particular device specifications.

b. The logical I/O drivers which perform data conversion and logical record packing.

c. The physical I/O drivers which communicate with the actual device.

All operations are performed on a record basis; i.e., card records contain 40 or less words. Any request processes exactly one record. If more words are requested than are contained in a record, only the amount contained in the record are significant.

5.2 LIO CALLING SEQUENCES

The LIO system accepts two different calling sequences:

1. Input/output request call.
2. Status check call.

Both are described in the following paragraphs. In the generalized calling sequences shown in these descriptions the label α identifies the branch instruction which transfers control to LIO. The labels of other elements are shown relative to α.

5.2.1 INPUT/OUTPUT REQUEST

In an assembly language program, the LIO user may request an input/output operation by programming the following calling sequence:

α	CALL	LIO
$\alpha+1$	DC	$/x_1x_2x_3x_4$
$\alpha+2$	DC	AREA
$\alpha+3$	DC	OPCOP
$\alpha+4$	(return location)	

where:

LIO — is the name of the entry to the LIO central command routine.

$/x_1x_2x_3x_4$ — represents a hexadecimal word (hexadecimal indicated by /) containing the following:

- x_1 — an LIO operation code as defined in table 5-1.
- x_2 — either 0 or 1 to specify data type-
  - 0 = ASCII data
  - 1 = Binary data
  - 2 = Special mode

x_3x_4 represents a logical unit number (hexadecimal) as defined in table 2-1.

NOTE

In the text which follows a reference to X_N is used specify the value of the subscripted hex field. For example; OPEN with X_2=1 specifies that hex digit two is to be set equal to 1(X1XX).

AREA represents the address of a data area. The value contained in the first word of the data area specifies the number of data words in the area. In symbolic notation this area is defined by the statements:

```
AREA    DC     n
        BSS    n
```

n = number of words in the data area.

OPCOP represents the address of a routine to be executed upon completion of the requested operation, or it is zero.

A zero indicates no user routine is provided.

Table 5-1. LIO Operation Code

CODE	OPERATION
0	No operation (ignored by LIO)
1	Read
2	Write
3	Write
4	Read
5	Open
6	Special for each device
7	Special for each device
8	Close
	An OPCOP address causes the user routine to be executed at the interrupt level (i.e., called by the I/O interrupt service routine when operation is complete). The user routine is executed as a subroutine and must accept the calling sequence. BSI OPCOP; DC loc_1. 'loc_i' contains the address of an I/O list (see section 6, I/O subroutines).

When this call is made to LIO, the operation is initiated and control is returned to the location following the DC OPCOP location.

5.2.2 I/O-REQUEST STATUS CHECK

If no OPCOP routine is specified in the I/O request calling sequence, the user may check the status of the operation with the call:

α	CALL	LIO
α +1	DC	/F$x_2x_3x_4$
α +2	(return location)	

where:

F	specifies a check operation
x_2	specifies the wait/no-wait option (0 or 1).
x_2=0	indicates that LIO should wait for the operation to be completed before returning to the user's program. If the device is not ready or an error condition exists, the operator is notified and may take remedial action. Upon return to the user's program, the A-register will contain the status of the operation as specified in table 5-2 (0, 1 or F only).
x_2=1	indicates that LIO is to place the current status of the operation in the A-register and return immediately. Any status code as specified in table 5-2 is possible.

x_3x_4 specifies the logical unit number (hexadecimal) of the unit being tested. Logical unit numbers are defined in section 2, table 2-1.

Return from this call is to the location immediately following $/Fx_2x_3x_4$.

Table 5-2. Status Indicators

CODE	STATUS
0	Operation ignored
1	Successful completion
2	Device off-line (logically)
3	Device not ready
4	Parity error (device dependent)
5	Write select (device dependent)
6	Data error (device dependent)
7	Data overrun (device dependent)
8	Seek error (device dependent)
9	File protect error (device dependent)
A	Bad sector address (device dependent)
B	Address modification (device dependent)
C	Unused
D	Unused
E	Unused
F	End-of-file
-1	Busy

5.3 LIO USAGE

Normal usage of the LIO system consists of using the status check and wait call as opposed to the OPCOP subroutine. Since the logical unit may be assigned to any physical device, the user should perform operations as follows:

a. Call LIO with an OPEN operation code. This insures that the device is initialized.

b. Call LIO to perform any number of read/write or special operations.

c. Call LIO with a CLOSE operation code to insure that all data has been processed.

The following subparagraphs describe the various logical I/O device drivers. Included, are calling sequences for each driver.

5.3.1 LOGICAL DISK DRIVER

The logical disk driver provides I/O processing using the following operation codes:

CODE	PROCESS
1,4	Read the specified number of words into the user.data area
2,3	Write the specified number of words on disk from the user data area.
5	Open - position at the beginning of a disk file.
A	Seek - position the disk at a specified sector

The user data area for disk I/O contains two parameters preceding the data area itself. The first specifies the number of words to be transmitted. The second specifies the logical sector number (relative sector position in the file) to be used in the transmission or seek operation.

An OPEN operation (5) will position the disk at the beginning of a file and set zero as the logical sector number.

Successive reads or writes will automatically process consecutive sectors. After each read or write operation, the logical sector number is adjusted to indicate the next available sector.

Logical I/O disk records (i.e., the user's data area) must be ≤320 words.

Following are disk I/O calling sequences (DK file is used; DP file could also be used):

1. OPEN - Position at beginning of disk file.

```
α       CALL    LIO     CALL LIO SYSTEM
α+1     DC      /510D   POSITION AT BEGINNING
                        OF DK FILE'
α+2     DC      A       IDENTIFY DATA AREA
α+3     DC      0       NO OPCOP
        (return location)
        .
        .
        .
A       DC      n       DEFINE RECORD SIZE
                        (n≤320)
        DC      (any)   THE VALUE WILL BE
                        ZEROES FOLLOWING OPEN
        BSS     n       RESERVE n WORDS.
```

2. SEEK - Position at specified sector.

```
α       CALL    LIO     CALL LIO SYSTEM
α+1     DC      /700A   SEEK SPECIFIED SECTOR
                        IN DK FILE.
α+2     DC      A       IDENTIFY DATA AREA.
α+3     DC      0       NO OPCOP
        (return location)
        .
        .
        .
A       DC      n       DEFINE RECORD SIZE
                        (n≤320)
        DC      s       DEFINE SECTOR NUMBER
                        (ALSO MAY BE ESTABLISHED
                        AS PRODUCT OF PROGRAM
                        EXECUTION)
        BSS     n       RESERVE n WORDS
```

3. READ

α	CALL	LIO	CALL LIO SYSTEM
α+1	DC	/110D	READ, BINARY, DK FILE
α+2	DC	A	IDENTIFY DATA AREA
α+3	DC	0	NO OPCOP
	(return location)		
	.		
	.		
	.		
A	DC	n	DEFINE RECORD SIZE ($n \leq 320$)
	DC	s	SECTOR NUMBER
	BSS	n	RESERVE DATA AREA

4. WRITE

α	CALL	LIO	CALL LIO SYSTEM
α+1	DC	/110D	READ, BINARY, DK FILE
α+2	DC	A	IDENTIFY DATA AREA
α+3	DC	0	NO OPCOP
	(return location)		
	.		
	.		
	.		
A	DC	n	DEFINE RECORD SIZE ($n \leq 320$)
	DC	s	SECTOR NUMBER
	BSS	n	RESERVE DATA AREA

5. CHECK STATUS

α	CALL	LIO	USAGE
α+1	DC	$/F\binom{0}{1}0D$	CHECK STATUS ON DK FILE
	(return location)		

5.3.2 LOGICAL DISK PACKING DRIVER

The logical disk packing driver performs packing and unpacking of data within an internal system buffer prior to performing disk operations. This driver processes the following operation codes:

CODE	PROCESS
1,4(read)	Read the number of words specified in the data area from the system buffer (<319 words).If the buffer is empty, initiate reading of the next sector into the buffer. If the number of words requested exceeds the length of the next record in the buffer, the driver will fill the remainder of the data area with blanks (ASCII) or zeros (binary).
2,3(write)	Write the number of words specified (<319)in the data area into the system buffer. If the buffer is full; write the buffer onto the disk, then transfer the data area to the buffer.
5(open)	Position at beginning of file (optionally attach user buffer).

CODE	PROCESS
8(close)	If the last operation was write; write the buffer onto disk, and position successive writes to start on the next available sector.

The disk files for these operations may be any packed file. (In the standard configuration, the only disk file that is not packed is ok.)

Before a series of read or writes using this driver, the OPEN operation must be performed. The series must be terminated by a CLOSE operation.

This driver provides a method for economic use of disk files by maintaining packed 320 word disk sectors. For example, when ASCII data are transmitted with this driver all excessive blanks are removed. Also, several records may occupy a single disk sector as opposed to the disk driver (5.3.1) where each read or write requires at least one sector. Unlike the disk driver, the disk packing driver does not require or use a sector number specification in the user data area.

User Buffers - The standard system configuration allocates one core buffer for system use. This results in very slow disk-to-disk packed transfers. The user may allocate some of his core for buffering (323 words). A user buffer may be shared by several files. To allocate a user buffer, OPEN with X_2=1.

Following are the calling sequences used with this driver (ASCII data and DP file specifications are used; however, binary data and working files may be used):

1. OPEN - Position at beginning of file.

α	CALL	LIO	CALL LIO
α+1	DC	$/5(^0_1)00$+LUN	OPEN, DP FILE
α+2	DC	UBUFF	ADDRESS OF USER BUFFER IF X_2=1 OR IGNORED
			TRANSMITTED
α+3	DC	0	NO OPCOP
	(return location)		
UBUFF	DC	0	IF A USER BUFFER IS DESIRED,
	DC	320	IT MUST BE OF THIS FORM
	BSS	321	

2. CLOSE - Terminate, write last buffer onto disk.

α	CALL	LIO	CALL LIO
α+1	DC	/800E	CLOSE, DP FILE
α+2	DC	any	NO EFFECT IN CLOSE OPERATION
α+3	DC	0	NO OPCOP
	(return location)		

3. READ

```
α       CALL    LIO       CALL LIO
α+1     DC      /100E     READ, ASCII, DP FILE
α+2     DC      A         IDENTIFY DATA AREA
α+3     DC      0         NO OPCOP
        (return location)
        .
        .
        .
A       DC      n         DEFINE AREA SIZE (n<319)
        BSS     n         RESERVE n WORDS
```

4. WRITE

```
α       CALL    LIO       CALL LIO
α+1     DC      /200E     WRITE, ASCII, DP FILE
α+2     DC      A         IDENTIFY DATA AREA
α+3     DC      0         NO OPCOP
        (return location)
        .
        .  .
        .
A       DC      n         DEFINE AREA SIZE (n<319)
        BSS     n         RESERVE n WORDS
```

5.3.3 LOGICAL CARD DRIVER

The logical card driver performs automatic conversion and buffering of data. The driver processes the following operation codes:

CODE	PROCESS
1,4 (read)	Read ASCII or binary data from the card read buffer into the specified data area. If the card contains a $ in column 1, an EOF status is returned.
2,3 (write)	Write ASCII or binary data from the specified data area to the card punch buffer. Initiate punching of the buffer.

OPEN and CLOSE operations on the card reader or punch have no effect.

The data area specified for card read/punch must be at least 40 words in length. Data is packed in this area as two card-columns per word (either ASCII or binary).

The card reader driver normally reads ahead two records. The user may inhibit this feature as follows:

```
REF         XEOF
SRA         16          Clear accumulator
STO    L    XEOF
```

1. READ

α	CALL	LIO	CALL LIO SYSTEM
α+1	DC	$/\binom{1}{4}\binom{0}{1}01$	READ, ASCII OR BINARY CARD READER
α+2	DC	A	IDENTIFY DATA AREA
α+3	DC	0	NO OPCOP
	(return location)		
	.		
	.		
	.		
A	DC	40	DEFINE RECORD SIZE
	BSS	40	RESERVE SPACE

α	CALL	LIO	CALL LIO SYSTEM
α+1	DC	$/\binom{2}{3}\binom{0}{1}02$	PUNCH, ASCII OR BINARY CARD PUNCH
α+2	DC	A	IDENTIFY DATA AREA
α+3	DC	0	NO OPCOP
	(return location)		
	.		
	.		
	.		
A	DC	40	DEFINE RECORD SIZE
	BSS	40	RESERVE SPACE

α	CALL	LIO	CALL LIO SYSTEM
α+1	DC	$/F\binom{0}{1}\binom{01}{02}$	CHECK, WAIT OR GIVE CURRENT STATUS, CARD DEVICE RETURN WITH INDICATOR IN A-REGISTER
		(return location)	
		.	
		.	
		.	

5.3.4 LOGICAL LINE PRINTER DRIVER

The logical line printer driver provides for both output of ASCII data and page formatting. It processes the following operation codes:

CODE	PROCESS
2,3 (write)	Output ASCII data onto one line on the printer followed by an upspace. Code 3 is used for FORTRAN writes (1st character form control).
5	OPEN - page eject. Position paper at top of next page.
7	Vertical page format. The contents of the area word specifies format controls as described below.

Control (Contents of Area Word)	LP Function
1	Immediate skip to channel 1
2	Immediate skip to channel 2
3	Immediate skip to channel 3
4	Immediate skip to channel 4
5	Immediate skip to channel 5
6	Immediate skip to channel 6
7	Immediate skip to channel 7
8	Immediate skip to channel 8
9	Immediate skip to channel 9
A	Immediate skip to channel 10
B	Immediate skip to channel 11
C	Immediate skip to channel 12
D	Immediate upspace of 1
E	Immediate upspace of 2
F	Immediate upspace of 3

Line Printer Format Control: The vertical formatting of printed output to the line printer (operation code 7) is guided by a carriage control tape on the line printer. The channels of this tape are used for vertical positioning of the paper. In addition, there are several immediate spacing methods which are not dependent on the tape.

Using LIO, the correspondence between the contents of the user's area word and format control is as follows:

At anytime that the bottom of the form is sensed on the printer, an automatic page eject is performed.

Calling Sequences:

1. OPEN - Page eject.

α	CALL	LIO	CALL LIO SYSTEM
α+1	DC	/5005	OPEN, ASCII, LINE PRINTER
α+2	DC	A	IDENTIFY DATA AREA (FOR PAGE EJECT' ANY AREA MAY BE USED)
α+3	DC	0	NO OPCOP
	(return location)		

2. WRITE - Print a line and upspace.

α	CALL	LIO	CALL LIO SYSTEM
α+1	DC	/$\binom{2}{3}$005	OUTPUT, ASCII, LINE PRINTER
α+2	DC	A	IDENTIFY DATA AREA
α+3	DC	0	NO OPCOP
	(return location)		
A	DC	n	DEFINE RECORD SIZE (n WORDS, TWO CHARACTERS PER WORD)
	BSS	n	RESERVE SPACE

3. FORMAT

	CALL	LIO	CALL LIO SYSTEM
	DC	/7005	FORMAT CONTROL, LINE PRINTER
	DC	F	IDENTIFIES WORD WHICH CONTAINS FORMATTING VALUE'
	DC	0	NO OPCOP
	(return location)		
	.		
	.		
	.		
F	DC	/000n	DEFINE FORMAT CONTROL WHERE n (F

4. CHECK STATUS

α	CALL	LIO	CALL LIO SYSTEM
α+1	DC	/F$\binom{0}{1}$05	CHECK, WAIT OR RETURN CURRENT STATUS, LINE PRINTER
	(return location)		

5.3.5 LOGICAL TELETYPE DRIVER

The logical teletype driver provides I/O operation functions for both the teletype keyboard/printer and teletype paper tape I/O. Selection between keyboard/printer and paper tape is a manual operation on the teletype unit.

This driver processes the following operation codes:

CODES	PROCESS
1,4 (read)	Input into specified data area.
2,3 (write)	Output from specified data area.

OPEN and CLOSE operations have no meaning to the teletype unit.

Teletype Binary Format: The first character of a binary paper tape record specifies the number of words in the record. The record may be up to 255 characters in length.

Input: On input the driver reads the first character to determine the record size in words and then reads the specified number of characters. The characters are formatted into words and stored into memory. If the input record is insufficient to fill the users area the remaining area is filled with zeroes. Leading zeroes are ignored.

Output: The record length is output as the first data word followed by the data record. Trailing zeroes are not output from the users area. Binary records always contain at least one word.

NOTE

All binary operations are executed with the teletype in the non-echo mode.

Teletype ASCII Format:

Input: When ASCII character input is requested, the driver first outputs a Line Feed, a ? and a Space and then accepts keyboard or paper tape input. The input characters are stored (2 characters per word) in the users' data area. Input terminates when a Return character is read. Leading zeroes (leader) and code deletes are ignored.

If the return occurs before the end of the data area, the remainder of the data area is filled with blanks. If the return occurs beyond the data area, the excess characters are ignored.

If a Rub-Out character is read in, input is reinitiated and all previously input characters are deleted.

If a back-arrow (←) character is read in, the last stored character in the data area is deleted. Any number of back-arrows may be read in, and with each, another previously stored character is deleted. The back-arrow characters are not counted as input characters and as each character is deleted, the input character count is decremented by one.

Output: A line feed and two spaces are output first followed by the characters in the data area. Trailing blanks in the data area are not output. A carriage return is output as the final character.

NOTE

The teletype is set to non-echo for binary input.

Teletype Character Format: This format consists of n 8-bit characters. Each word in the data area contains two characters. Use a subfunction code of $2(X_2=2)$. The first word of the data area must be the character count.

Input: Exactly n characters are input. Odd numbered characters fill bits 0-7 and even numbered characters fill bits 8-15.

Output: Exactly n characters are output.

Calling Sequences

1. READ

```
a       CALL      LIO              CALL I/O SYSTEM
                    0
                     1
a+1     DC        /1( )00+lun      READ
                     2
a+2     DC        A                DATA AREA
a+3     DC        0                NO OPCOP
        (RETURN LOCATION)

A       DC        N                AREA SIZE (WORDS OR CHARACTERS)
        BSS       n                AREA
```

2. PUNCH

```
        CALL      LIO              CALL LIO SYSTEM
                    0
                     1
a+1     DC        /2( )00+lun      PUNCH
                     2
a+2     DC        A                DATA AREA
        (RETURN LOCATION)

A       DC        n                AREA SIZE (WORDS OR CHARACTERS)
        BSS       n                AREA
```

3. CHECK STATUS

```
a       CALL      LIO              CALL LIO SYSTEM
                    0
a+1     DC        /F(1)00+lun      CHECK, WAIT OR GIVE CURRENT STATUS
        (RETURN LOCATION)
```

4. PUNCH LEADER (Open or Close) Call LIO system

```
a       CALL      LIO
                    5
a+1     DC        (8)000+lun       (CLOSE WAITS FOR COMPLETION PRIOR TO RETURN TO USER)
a+2     DC        **               NOT USED
a+3     DC        0                NO OPCOP
```

5. PUNCH

```
a       CALL      LIO              CALL LIO SYSTEM
a+1     DC        /800X            CLOSE, PAPER TAPE
a+2     DC        ANY              NO EFFECT
a+3     DC        O                NO OPCOP
```

5.3.6 LOGICAL PAPER TAPE DRIVERS

The logical paper tape drivers provide I/O operation functions for the paper tape reader and paper tape punch.

This driver processes the following operation codes:

CODES	PROCESS
1,4(read)	Input into specified data area.
2,3(write)	Output from specified data area.
5,(open)	No effect on paper tape reader. Punch 10 inch leader on paper tape when punch is specified.
8(close)	No effect on paper tape reader. Punch 10 inch trailer on paper tape when punch is specified. A close waits for completion prior to return to user.

Paper Tape Binary Format: The first character of a binary paper tape record specifies the number of words in the record. The record may be up to 255 characters long.

Input: On input the driver reads the first character to determine the record size in words (leading zeroes are bypassed) and then reads the specified number of characters and stores them, two characters per word, into the user's data area. The remainder of the user area is set to zero.

Output: Trailing zeroes are deleted from output records. The record length (in words) is output preceding the record. Binary records will always contain at least one word.

Paper Tape ASCII Format

Input: The input characters are stored (2 characters per word) in the user's data area. Input terminates when a Return character is read. Leading zeroes (leader) are ignored. Line feed characters are ignored.

If the return occurs before the end of the data area, the remainder of the data area is filled with blanks. If the return occurs beyond the data area, the excess characters are ignored.

If a Rub-Out character is read in, input it reinitiated and all previously input characters are deleted.

If a back-arrow (←) character is read in, the last stored character in the data area is deleted. Any number of back-arrows may be read in, and with each, another previously stored character is deleted. The back-arrow characters are not counted as input characters and as each character is deleted, the input character count is decremented by one.

Output: A line feed is output first, followed by the characters in the data area. Trailing blanks are not output. A return is output at the end of the record.

Paper Tape Character Format: This character format consists of n 8-bit characters. Each word in the data area contains two characters. Also a subfunction code of 2. The first word of the data area is a character count.

Input: Exactly n characters are input. Character one fills bits 0-7 and character two bits 8-15.

Output: Exactly n characters are output.

Calling Sequences

1. READ

α	CALL	LIO_0	CALL LIO SYSTEM
α+1	DC	$1(^1_2)00$+lun	READ
α+2	DC	A	IDENTIFY DATA AREA
α+3	DC	0	NO OPCOP
	(RETURN LOCATION)		
A	DC	n	DEFINE AREA SIZE (WORDS OR CHARACTERS)
	BSS	n	

2. PUNCH

α	CALL	LIO_0	CALL LIO SYSTEM
α+1	DC	$/2(^1_2)00$+lun	PUNCH
α+2	DC	A	IDENTIFY DATA AREA
α+3	DC	0	NO OPCOP
	(RETURN LOCATION)		
A	DC	n	DEFINE AREA SIZE (WORDS OR CHARACTERS)
	BSS	n	RESERVE n WORDS

3. CHECK STATUS

```
α       CALL    LIO           CALL LIO SYSTEM
                  0
α+1     DC      /F(1)00       CHECK, WAIT OR GIVE CURRENT STATUS
        (RETURN LOCATION)
```

4. PUNCH LEADER OR TRAILER (Open or Close)

```
α       CALL    LIO           CALL LIO SYSTEM
                 5
α+1     DC      (8)000+lun    (CLOSE WAITS FOR COMPLETION)
α+2     DC      ANY           NO EFFECT
α+3     DC      0             NO OPCOP
```

NOTE

A close insures all data is fully output from packed buffers.

SECTION 6
DESCRIPTION OF I/O SUBROUTINES

6.1 GENERAL

All input/output operations in DBOS are performed by subroutines referred to as I/O drivers. The I/O drivers perform the function of formatting and transferring data to and from the various peripheral devices in a system. Each peripheral device is supported by a unique I/O driver subroutine which performs all the required formatting, error recovery and interrupt processing for that device.

The I/O drivers supplied with DBOS are used by the logical I/O system (LIO) for all data transfer operations. In addition, the disk I/O subroutine may be referenced directly by the user for special I/O requirements. All required I/O drivers are contained in the resident monitor. All I/O drivers contained within DBOS are coded as non re-entrant subroutines. That is, one operation must be completed before a subsequent operation is initiated. When successive calls are made to the same I/O driver, the driver will wait internally for the previous operation to be completed before the later request is initiated and control is returned to the user. There is no queueing of I/O requests.

6.1.1 I/O DRIVER ORGANIZATION

All I/O drivers are organized into two portions - an I/O initialization and an interrupt response routine.

I/O Initialization Routine: The functions of the I/O initialization routine are to verify the I/O list presented by the user, determine device readiness to accept a command and build a data pool for use by the interrupt response routine. Immediately upon entry, the I/O initialization routine determines if the previously initiated operation is complete. If not, a wait loop is entered until such time as the prior operation is completed. Once the I/O driver is free to process another request, the control parameter is interrogated to determine if the operation code is valid (see basic calling sequence, paragraph 6.2, for a complete description of the I/O list). If the control parameter contains an operation code (X_1) which is not valid for the I/O driver referenced, the error parameter is set with an operation complete indication (1) and control is returned to the user immediately. Following the verification of the control parameter, both the device and I/O channel are checked to determine if an I/O operation can be initiated. If the device returns a not ready response, the error parameter in the list is set to three (3) and an exit is made to the user immediately. An indication that the I/O channel is presently being used by another device will cause the initialization routine to loop until the channel becomes free. The convention adopted is that the even channel register must contain zero (0) before an operation is initiated. All I/O drivers write zero into the register upon final completion of an operation.

When all of the forementioned checks have been made, the initialization routine will set the link/busy indicator to minus one (1), initiate the operation and return control to the user. Control will be returned to the I/O driver interrupt response routine upon occurrence of an interrupt to signal completion of the operation.

Interrupt Response Routine: The interrupt response routines are entered as a result of an I/O interrupt. Upon interrupt, control is initially transferred to the appropriate interrupt level processor which in turn transfers control to the appropriate I/O driver interrupt routine. The interrupt routine checks for errors, performs data formatting, initiates subsequent operations for character oriented devices and sets I/O list parameters when a request is completed. Whenever possible erroneous operations are retried a specified number of times. If errors persist or the device is not capable of error recovery, the user is notified of the type of error via the error parameter in the I/O list.

6.2 BASIC CALLING SEQUENCE

All I/O drivers are entered via a standard calling sequence. The calling sequence for BULKN follows the basic pattern for all drivers, but is the only driver to which the user may have access. A special indirect address vector is provided for this access. The following is the format of the calling sequence:

```
        CALL    NAME        (I/O DRIVER NAME)
        DC      LIST        ADDRESS OF I/O LIST
        .
        .
        .
LIST    DC      0           LINK/BUSY
        DC      0           OP-COP (0 OR SUBROUTINE ADDRS)
        BSS     4           4 WORDS OF SYSTEM RESERVED
        DC      0           ERROR PARAMETER
        DC      /x1x2x3x4   CONTROL PARAMETER
        DC      AREA        ADDRES OF I/O AREA
        .
        .
        .
AREA    DC      WORDS       WORD COUNT
        BSS     Words       BUFFER AREA
```

6.2.1 NAME PARAMETER

The NAME PARAMETER is the symbolic name of the I/O driver. The reference name for the indirect vector to BULKN is BULKA.

6.2.2 I/C LIST PARAMETERS

All calling sequences have nine (9) I/O list parameters. The I/O list conveys to the driver all the information required to perform an I/O operation.

Link/Busy: The LINK/BUSY indicator is used by a calling program to determine when the operation requested of the driver is complete. When an operation is in progress, the LINK/BUSY parameter will contain either a positive or negative value. Upon completion of the call, the indicator is set to zero. It is the responsibility of the user to make certain that a previous operation performed through the I/O list is complete before a subsequent call is made to a driver using the same list.

OPCOP: The OPCOP parameter may be used to specify the address of a subroutine to be entered when the operation requested by the I/O list is complete. If the parameter contains a zero, no OPCOP subroutine entry is specified.

If the value of the parameter is non-zero, the driver assumes the value to be the entry address to a subroutine (entered via a BSI instruction). The parameter sent to the OPCOP subroutine is the address of a word which contains the list address whose operation has just been completed. Entry to the user OPCOP routine is made with a calling sequence as shown:

BSI	L	USER	(OPCOP subroutine address)
DC		PARA	Address of list parameter
	.		
	.		
	.		
PARA	DC	LIST	Contains address of list completed

The user may again call the I/O driver from within the OPCOP routine, only if all of the following are true:

1. The error parameter in the list is set to a one (1), (operation complete).
2. The control parameter specified a valid op-code on the previous request, and
3. No request was made to the I/O driver between the time of the initial call and the OPCOP exit.

Also, if a subsequent call is made to an I/O driver from within the OPCOP routine, no status waits or delays can be used since the OPCOP routine is entered in an interrupted state.

System Reserved: These four (4) words of the I/O list are required for compatability with MPX calling sequences.

Error Parameter: The seventh list parameter is the error parameter. When the link/busy indicator is set to zero, the error parameter is set to one of the following values:

1. Successful completion of the call.
2. Device logically disconnected from the system.
3. Device hardware not-ready.

   Any hardware malfunction which did not allow completion of the call.

Error values of greater than 3 generally have different meanings for each driver.

Note that it is the user's responsibility to initiate any desired error recovery procedures.

Control Parameter: The eighth list parameter is the CONTROL PARAMETER. This parameter consists of four hexadecimal digits which define the I/O operation. The first digit defines the I/O function in terms of read, write, control, etc. The second digit modifies the function digit. The ninth and final list parameter is the address of the I/O area (AREA).

Area: AREA is the label of the users I/O area.

Error Parameter: The seventh list parameter is the error parameter. When the link/busy indicator is set to zero, the error parameter is set to one of the following values:

1. Successful completion of the call.
2. Device logically disconnected from the system.
3. Device hardware not-ready.

>3 Any hardware malfunction which did not allow completion of the call.

Error values of greater than 3 generally have different meanings for each driver.

Note that it is the user's responsibility to initiate any desired error recovery procedures.

Control Parameter: The eighth list parameter is the CONTROL PARAMETER. This parameter consists of four hexadecimal digits which define the I/O operation. The first digit defines the I/O function in terms of read, write, control, etc. The second digit modifies the function digit. The ningh and final list parameter is the address of the I/O area (AREA).

Area: AREA is the label of the users: I/O area.

6.3 GENERAL FORMAT OF I/O CALLS

6.3.1 CALLING SEQUENCES

The general format for assembler language calls with a type one (1) or type two (2) exit is as follows:

```
        CALL        NAME        I/O DRIVER NAME
        DC          LIST        POINTS TO I/O LIST
        LD          LIST        BUSY TEST
        BSC    L    *-3, Z
        MDX    L    LIST+6, -1  DETERMINE IF I/O OPERATION
        BSI         ERROR       COMPLETED SUCCESSFULLY
        NOP                     BRANCH TO ERROR IF NOT,
                                OTHERWISE CONTINUE
        .
        .
        .
LIST    DC          0           LINK/BUSY
        DC          0           EXIT TYPE (0 OR SUBR ADDR)
        BSS         4           SYSTEM RESERVED 1 to 4
        DC          0           ERROR PARAMETER
        DC          /XXXX       CONTROL PARAMETER
        DC          AREA        I/O AREA ADDRESS
        .
        .
        .
AREA    DC          WDCT        WORD COUNT
        BSS         WDCT        DATA AREA
```

6.4 BULK STORAGE SUBROUTINE (BULKN)

The bulk storage subroutine performs all reading and writing of data relative to the model 1341 and 1344 disk storage unit. This includes the major functions; seek, read and write in conjunction with read-back check.

BULKN reads and writes consecutive sectors most of the time (depending on when the disk interrupt occurs) on most systems without extra disk revolutions. Successful use of the bulk storage subroutines can be expected only if programs are built within the framework of certain conventions. The primary concern behind the convention is the safety of data recorded on the disk. The file-protection scheme is dependent upon the sector-numbering technique. It contributes to data integrity by allowing the disk subroutine to verify the correct positioning of the access arm before it actually performs write operations. This verification requires that sector identifications be pre-recorded on each sector and that subsequent writing to the disk be done in a manner that preserves the existing identification. The disk subroutines have been organized to comply with these requirements. The sector numbers are recorded at system generation time.

Sector Numbering: The details of the numbering scheme are as follows: Each disk sector is assigned a logical address from the sequence 0,1...,8104 for 1341 (1608 for model 1344 and 1345) corresponding to the sector's position in the ascending sequence of cylinder and sector numbers from cylinder 0 (outermost) sector 0, through cylinder 202 (innermost) sector 39 for 1341 (sector 7 for model 1344). An additional four tracks are used as alternates.

Utilization of this first word for identification purposes diminishes the per sector availability of data words to 320; therefore, transmission of full sectors of data is performed in units of this amount.

Calling Sequence

```
        REF        BULKA
        BSI    I   BULKA
        DC         LIST
        .
        .
LIST    DC         0            LINK/BUSY
        DC         OPCOP        0 IF NOT TYPE 2 EXIT
        BSS        4            SYSTEM RESERVED 1 to 4
        DC         0            ERROR PARAMETER
        DC         /XXXX        CONTROL PARAMETER
        DC         AREA         I/O AREA ADDRESS
        .
        .
        .
AREA    DC         WDCT         WORD COUNT
        DC         SECAD        SECTOR ADDRESS
        BSS        WDCT         DATA AREA
        .
        .
        .
OPCOP   DC         0            OP-COMPLETE SUBROUTINE
                                ENTRY POINT
        .
        .
        .
        MDX    L   OPCOP,1      SET UP RETURN ADDRESS
        BSC    I   OPCOP        EXIT BACK TO IOCR
```

List Parameters: Link/Busy. Upon completion of the I/O call specified by the list, this parameter is set to zero. Link/Busy must be 0,/4400 or /4480 at the time the driver is called.

Exit Type: If zero, this parameter indicates a type one (1) or type three (3) exit is to be made. If non-zero, it indicates a type two (2) exit is to be made and it contains the entry address of the operation-complete subroutine.

System Reserved 1-4: These words are reserved for system use only. See basic calling sequence.

Error Parameter: This parameter is set upon I/O completion to one of the following values:

Value	Meaning
1	Successful completion of call
2	Device logically off-line
3	Device not-ready
4	Parity error
5	Write select
6	Data error
7	Data overrun
8	Seek error

Control Parameter: This parameter consists of four hexadecimal digits as defined below.

Hexadecimal Digit 1. This digit defines the I/O operation and must be set to one of the following values:

Value	Meaning
0	Put device in on-line or off-line status (see hexadecimal digit 2).
1	Read. Positions the access arm and reads data into the user's I/O area until the specified number of words have been transmitted. Although sector identification words are read and checked for agreement with expected values, they are neither transmitted to the I/O data area nor are they counted in the tally of words read. If during the reading of a sector a read check occurs, the operation is retried a maximum of 15 times. If the error persists, the function is discontinued, and the error parameter is set in the I/O list.

Value	Meaning
2	Write without readback check. The function is the same as write with readback check, except that no readback check is performed.
3	Write with readback check. This function writes the contents of the indicated I/O data area into consecutive disk sectors. Writing begins at the designated sector and continues until the specified number of words has been transmitted. A readback check is performed on the data written. If any errors are detected, the operation is retried a maximum of 15 times. If the function cannot be completed in 15 tries, the ERROR PARAMETER is set in the I/O LIST.
4	Write immediate. Writes data with no attempt to check for hardware errors. This function is provided to fulfill the need for more rapid writing to the disk than is provided in the previously described write function. The primary application of write immediate is in the 'streaming' of analog

Value	Meaning
4 (continued)	input data to the disk for temporary bulk storage.
5	Seek. Initiates a seek as specified by the seek option digit. If any errors are detected, the operation is retried a maximum of 15 times.

Hexadecimal Digit 2. When digit 1 is zero (0), this digit specifies whether the device is to be put on-line or taken off-line.

0 - take device off-line

1 - put device on-line

If digit 1 specifies a seek (function code 5) then digit 2 specifies the seek option. If zero, a seek is executed to the cylinder whose sector address is in the disk I/O area control word. If non-zero, a seek is executed to the next cylinder toward the center, regardless of the sector address in the disk I/O area control word. The seek option is valid only when the seek function is specified.

NOTE

The seek function requires that the user set up the normal I/O area used. The I/O area control word (first word) is ignored.

Area Parameter: The I/O area parameter is the address of the first word of the user's I/O area. The first word contains a count of the number of data words that are to be transmitted during the disk operation. This count need not be limited by sector or cylinder size, since the BULKN subroutine crosses sector and cylinder boundaries, if necessary, in order to process the specified number of words.

The second word contains the sector address where reading or writing is to begin.

Following the two control words is the users data area. No chaining of disk I/O area is permitted.

Operation-Complete Subroutine: There is one parameter passed to the user's operation-complete subroutine and that is the address of the list most completed.

APPENDIX A

DISK SECTOR MAP

FILE	ALLOCATED SECTORS	EFFECTIVE SIZE
DC	100 - 2FF	2560 binary cards
LB	300 - 4FF	2560 binary cards
WC	500 - 57F	640 binary cards
WB	580 - 5FF	640 binary cards
WS	600 - 8FF	11,500[1] ASCII cards
DS	900 - FFF	27,000[1] ASCII cards
UL	1000 - 10FF	1,280 binary cards
DP	1100 - 11FF	1,280 binary cards or 4,000[1] ASCII cards
DK	1200 - 12FF	256 sectors
DJ	1300 - 13FF	3840 ASCII cards

Sectors 0 - FF are for system use only.

Sectors 1400 - 1F3F are available to the user.

[1] Approximate number of cards.

APPENDIX B
STANDARD CHARACTER CODES

The ASCII code is an eight-bit code, represented in this table by hexadecimal digits. Although the printer recognizes a six-bit code (that is, ignores the two high-order bits of the eight-bit code), the codes are represented in this table by hexadecimal digits.

Graphic or Control	ASCII (Hexadecimal)	Display Hollerith	Printer (Hexadecimal)
NULL	80		
SOM	81	—	
EOA	82		
EOM	83		
EOT	84		
WRU	85		
RU	86		
BELL	87	Bell	
FE	88	Cursor ←	
H. Tab	89	TAB	
Line Feed	8A	Line Feed	
V. Tab	8B		
Form	8C	Form Feed	
Return	8D	CR	
SO	8E	~~Format~~ An	erlaube Sonderz. (3)
SI	8F	—	
DCO	90	erlaube Sonderzeich.	
X-On	91	X-On	
Tape Aux. On	92	–	
X-Off	93	–	
Tape Aux. Off	94	–	
Error	95	–	
Sync	96	–	
LEM	97	SOM	
S0	98	–	
S1	99	Cursor →	
S2	9A	Cursor ↑	
S3	9B		
S4	9C	Variablen End	
S5	9D	Variablen Start	
S6	9E	Blink End	
S7	9F	Blink Start	

Graphic or Control	ASCII (Hexadecimal)	Hollerith IBM 029*	Hollerith IBM 026	Printer (Hexadecimal)
ACK	FC			
Alt. Mode	FD			
Rubout	FF			
!	A1	5-8	11-2-8	E1
"	A2	7-8	0-5-8	E2
#	A3	3-8	0-7-8	E3
$	A4		11-3-8	E4
%	A5	0-4-8	11-7-8	E5
&	A6	12	12-7-8	E6
'	A7	5-8	4-8	E7
(	A8	12-5-8	0-4-8	E8
)	A9	11-5-8	12-4-8	E9
*	AA		11-4-8	EA
+	AB	12-6-8	12	EB
,	AC		0-3-8	EC
-	AD		11	ED
.	AE		12-3-8	EE
/	AF		0-1	EF
:	BA	2-8	5-8	FA
;	BB		11-6-8	FB
<	BC	12-4-8	12-6-8	FC
=	BD	6-8	3-8	FD
>	BE	0-6-8	6-8	FE
?	BF	0-7-8	12-2-8	FF
[	DB		12-5-8	DB
\	DC		0-6-8	DC
]	DD		11-5-8	DD
↑	DE		7-8	DE
←	DF		2-8	DF
@	C0	4-8	0-2-8	C0
blank	A0		No Punch	E0
0	B0		0	F0
1	B1		1	F1
2	B2		2	F2
3	B3		3	F3
4	B4		4	F4
5	B5		5	F5
6	B6		6	F6
7	B7		7	F7

* MUST BE USED WITH DBOS

Graphic or Control	ASCII (Hexadecimal)	Hollerith	Printer (Hexadecimal)
8	B8	8	F8
9	B9	9	F9
A	C1	12-1	C1
B	C2	12-2	C2
C	C3	12-3	C3
D	C4	12-4	C4
E	C5	12-5	C5
F	C6	12-6	C6
G	C7	12-7	C7
H	C8	12-8	C8
I	C9	12-9	C9
J	CA	11-1	CA
K	CB	11-2	CB
L	CC	11-3	CC
M	CD	11-4	CD
N	CE	11-5	CE
O	CF	11-6	CF
P	D0	11-7	D0
Q	D1	11-8	D1
R	D2	11-9	D2
S	D3	0-2	D3
T	D4	0-3	D4
U	D5	0-4	D5
V	D6	0-5	D6
W	D7	0-6	D7
X	D8	0-7	D8
Y	D9	0-8	D9
Z	DA	0-9	DA

APPENDIX C
EBCDIC DECIMAL EQUIVALENCE

These values may be used by the Fortran programmer for character checking.

	FILL	NICOMP
	EBCDIC Character	Decimal Equivalence
Low	(12-0)	-16320
Listed in Collating Sequence	A	-16064
	B	-15808
	C	-15552
	D	-15296
	E	-15040
	F	-14784
	G	-14528
	H	-14272
	I	-14016
	(11-0)	-12224
	J	-11968
	K	-11712
	L	-11456
	M	-11200
	N	-10944
	O	-10688
	P	-10432
	Q	-10176
	R	-9920
	S	-7616
	T	-7360
	U	-7104
	V	-6848
	W	-6592
	X	-6336
	Y	-6080
	Z	-5824

	FILL EBCDIC Character	NICOMP Decimal Equivalence
Low	0	-4032
	1	-3776
	2	-3520
	3	-3264
	4	-3008
	5	-2752
	6	-2496
	7	-2240
	8	-1984
	9	-1728
	blank	16448
	(period)	19264
	<(less than)	19520
	(	19776
	+	20032
	&	20544
	$	23360
	*	23616
	)	23872
	-(minus)	24640
	/	24896
	,	27456
	%	27712
High	#	31552
	@	31808
	'(apostrophe)	32064
	=	32320

Listed in Collating Sequence

APPENDIX D
FORTRAN EXECUTION (RUNTIME) ERRORS

FORTRAN execution errors are classified into two categories; those which cause a run to be aborted and those which are logged but processing is allowed to continue.

In either case a message is output to OM which identifies the error, the subroutine and associated variables. This message takes the form:

**ERROR NUMBER IN NAME UP TO 8 VALUES

Table D-1 details the error message and system affect for each logable error.

Table D-1

No.	In Routine	Abort	Values	Reason
1	COMGO	No	Variable	Variable range error
1	LUC$	Yes	-	Conflict in logical unit usage
1	BCKSP	Yes	-	Bad binary record
1	DSKIO	Yes	File No.	File not defined
2	DSKIO	Yes	-	No files defined
3	DSKIO	Yes	-	Too many arguments
4	DSKIO	Yes	Rec.No	Illegal record number
1	FRMAT	No	Format address, buffer pointer, terminal character	Buffer exceeded on input (the variables listed are output for all FRMAT errors).
2	FRMAT	No	Same as 1 FRMAT	Buffer exceeded on output
3	FRMAT	No	Same as 1 FRMAT	Input exponent >99

Table D-1. (continued)

No.	In Routine	Abort	Values	Reason
4	FRMAT	No	Same as 1 FRMAT	Real data in integer field
5	FRMAT	No	Same as 1 FRMAT	Integer too large on input
6	FRMAT	No	Same as 1 FRMAT	Exponent overflow on input
7	FRMAT	No	Same as 1 FRMAT	Exponent underflow on input
16	FRMAT	Yes	Same as 1 FRMAT	Argument has no format
1	UNFMT	Yes	Data count, block count	Too many arguments

APPENDIX E
FORTRAN COMPILATION ERRORS

All FORTRAN compilation errors cause an abort situation. No binary output is generated and any attempt to build a program subsequently will result in a Processing Error.

Table E-1
FORTRAN Error Codes

Error Number	Cause of Error
C1	Non-numeric character in statement number.
C2	More than five continuation cards, or continuation card out of sequence.
C3	Syntax error in CALL LINK or CALL EXIT statement or END statement missing.
C4	Undeterminable, misspelled, or incorrectly formed statement.
C5	Statement out of sequence.
C6	Statement following STOP, RETURN, CALL LINK, CALL EXIT, GO TO, IF, does not have statement number.
C7	Name longer than five characters, or name not starting with an alphabetic character.
C8	Incorrect or missing subscript within dimension information (DIMENSION, COMMON, REAL, or INTEGER).
C9	Duplicate statement number.
C10	Syntax error in COMMON statement.

Table E-1. (continued)

Error Number	Cause of Error
C11	Duplicate name in COMMON statement.
C12	Syntax error in FUNCTION or SUBROUTINE statement.
C13	Parameter (dummy argument) appears in COMMON statement.
C14	Name appears twice as a parameter in SUBROUTINE or FUNCTION statement.
C16	Syntax error in DIMENSION statement.
C17	Subprogram name in DIMENSION statement.
C18	Name dimensioned more than once, or not dimensioned on first appearance of name.
C19	Syntax error in REAL, INTEGER, or EXTERNAL statement.
C20	Subprogram name in REAL or INTEGER statement.
C21	Name in EXTERNAL which is also in a COMMON or DIMENSION statement.
C22	IFIX or FLOAT in EXTERNAL statement.
C23	Invalid real constant.
C24	Invalid integer constant.
C25	More than 15 dummy arguments, or duplicate dummy arguments in statement function argument list.
C26	Right parenthesis missing from a subscript expression.
C27	Syntax error in FORMAT statement.
C28	FORMAT statement without statement number.
C29	Field width specification greater than 145.

Table E-1. (continued)

Error Number	Cause of Error
C30	In a FORMAT statement specifying E or F conversion, w greater than 127, d greater than 31, or d greater than w, where w is an unsigned integer constant specifying the total field length of the data and d is an unsigned integer constant specifying the number of decimal places to the right of the decimal point.
C31	Subscript error in EQUIVALENCE statement.
C32	Subscripted variable in a statement function.
C33	Incorrectly formed subscript expression.
C34	Undefined variable in subscript expression.
C35	Number of subscripts in a subscript expression does not agree with the dimension information.
C36	Invalid arithmetic statement or variable; or, in a FUNCTION subprogram, the left side of an arithmetic statement is a dummy argument (or in COMMON).
C37	Syntax error in IF statement.
C38	Invalid expression in IF statement.
C39	Syntax error or invalid simple argument in CALL statement.
C40	Invalid expression in CALL statement.
C41	Invalid expression to the left of an equal sign in a statement function.
C42	Invalid expression to the right of an equal sign in a statement function.
C43	If an IF, GO TO, or DO statement, statement number is missing invalid, incorrectly placed, or is the number of a FORMAT statement.
C44	Syntax error in READ or WRITE statement.

Table E-1. (continued

Error Number	Cause of Error
C46	FORMAT statement number missing or incorrect in a READ or WRITE statement.
C47	Syntax error in input/output list; or an invalid list element; or, in a FUNCTION subprogram, the input list element is a dummy argument or in COMMON.
C48	Syntax error in GO TO statement.
C49	Index of a computed GO TO is missing, invalid, or not preceded by a comma.
C51	Incorrect nesting of DO statements; or the terminal statement of the associated DO statement is a GO TO, IF, RETURN, FORMAT, STOP, PAUSE or DO.
C52	More than 25 nested DO statements.
C53	Syntax error in DO statement.
C54	Initial value in DO statement is zero.
C55	In a FUNCTION subprogram the index of DO is a dummy argument or in COMMON.
C56	Syntax error in BACKSPACE statement.
C57	Syntax error in REWIND statement.
C58	Syntax error in END FILE statement.
C59	Syntax error in STOP statement or STOP statement in process program.
C60	Syntax error in PAUSE statement.
C61	Integer constant in STOP or PAUSE statement is greater than 9999.

Table E-1. (continued)

Error Number	Cause of Error
C62	Last executable statement before END statement is not a STOP, GO TO, IF, CALL EXIT or RETURN.
C63	Statement contains more than 15 different subscript expressions.
C64	Statement too long to be scanned due to compiler expansion of subscript expressions or compiler addition of generated temporary storage locations.
C65*	All variables are undefined in an EQUIVALENCE list*
C66*	Variable made equivalent to an element of an array, in such a manner as to cause the array to extend beyond the origin of the COMMON area*
C67*	Two variables or array elements in COMMON are equated, or the relative locations of two variables or array elements are assigned more than once (directly or indirectly)*
C68	Syntax error in a EQUIVALENCE statement; or an illegal variable name in an EQUIVALENCE list.
C69	Subprogram does not contain a RETURN statement, or a main-line program contains a RETURN statement.
C70	No DEFINE FILE in a mainline program which has disk READ, WRITE or FIND statements.
C71	Syntax error in DEFINE FILE.
C72	Duplicate DEFINE FILE, more than 75 DEFINE FILES, or DEFINE FILE in subprogram.
C73	Syntax error in record number of READ, WRITE, or FIND statement.
C74	INSKEL COMMON referenced with two word integers.

Table E-1. (continued)

Error Number	Cause of Error
C75	Syntax error in data statement.
C76	Names and constants in a data statement not one to one.
C77	Mixed mode in data statement.
C78	Invalid hollerith constant in a data statement.
C79	Invalid hexadecimal specification in a data statement.
C80	Variable in a data statement not used elsewhere in the program.
C81	Common variable loaded with a data specification.
C82	Data statement too long.

* The decision of a code 65, 66 or 67 error prevents any subsequent detection of any of these three errors.

APPENDIX F
DBOS LOGICAL UNIT ASSIGNMENTS

NUMBER	NAME	USAGE	STANDARD ASSIGNMENT (FILE) (SEE FILE NAMES, TABLE 2-2)
0	CC	Control Command Input	TY (Teletype Keyboard)
1	SI	Symbolic Input	CR (Card Reader)
2	SO	Symbolic Output	WS (Working Symbolic File)
3	BI	Binary Input	WB (Working Binary File)
4	BO	Binary Output	WB (Working Binary File)
5	LO	Listing Output	LP (Line Printer)
6	IS	Intermediate Symbolic	WS (Working Symbolic File)
7	OM	Operator Messages	TY (Teletype Printer)
8	CI	Core Image Data	WC (Working Core Image File)
9	LB	Binary Library	LB (Directoried Library File)
10	SL	System Log	LP (Line Printer)
12	SB	Secondary Binary Library	CR (Card Reader)
11	UL	User Library	UL (Directoried User Library File)
13		User Disk Temporary	DK (Unformatted Disk I/O File)
14		User Packed Disk Temporary	DP (Disk)
15		NO	NO (Delete I/O)

APPENDIX G
DBOS FILE NAMES AND DESCRIPTION

NAME	DEVICE	USAGE
DS[1]	Disk	Directoried source language program data
LB[2]	Disk	Directoried binary object subroutine library
UL[2]	Disk	Directoried user binary object subroutine library.
DC[3]	Disk	Directoried core image program data
DJ[1]	Disk	Directoried job string file
WS[1]	Disk	Working source language data
WB[2]	Disk	Working binary object data
WC[2]	Disk	Working core image data
CR	Card Reader	ASCII or binary card input
CP	Card punch	ASCII or binary card output
LP	Line Printer	ASCII listing output
TY	Teletype	ASCII or binary teletype input/output
PR	Paper Tape Reader	ASCII or binary high-speed paper tape input
PP	Paper Tape Punch	ASCII or binary high-speed paper tape output
DK[3]	Disk	ASCII or binary disk sector input/output
DP[2]	Disk	ASCII or binary logical packed disk input/output
NO	None	Delete input or output

1 ASCII character string 80 character records. Data type A in file manipulation commands.

2 Binary object format 54 word records. Data type B in file manipulation commands.

3 Binary data in 320 word record form.