SERIES 60 (LEVEL 6)

GCOS 6 MOD 400 PROGRAM EXECUTION AND CHECKOUT ADDENDUM A

SUBJECT

Changes and Additions to the Manual

SPECIAL INSTRUCTIONS

Insert attached pages into Revision 0 of the manual dated November 1977 (see Collating Instructions). Except in completely revised Section 6, change bars indicate new and changed information and asterisks denote deletions.

NOTE: Insert this addendum cover behind the manual cover to indicate that the manual is updated with Addendum A.

SOFTWARE SUPPORTED

This update supports Release 0110 of the Series 60 (Level 6) GCOS 6 MOD 400 software system. For any later release of MOD 400 software, see the Manual Directory of the latest *System Concepts* manual to ascertain whether this update supports that release.

ORDER NUMBER

CB21A, Rev. 0

June 1978

21004
3678
Printed in U.S.A.

Honeywell

COLLATING INSTRUCTIONS

To update this manual, remove old pages and insert new pages as follows:

Remove	Insert
iii/blank	iii/blank
v/vi	v/vi
vii/blank	vii/blank
2-1/2-2	2-1/2-2
2-5/2-6	2-5/2-6
2-7/2-8	2-7/2-8
2-9/2-10	2-9/2-10
2-11/2-12	2-11/2-12
2-15/2-16	2-15/2-16
2-17/2-18	2-17/2-18
2-19/2-20	2-19/2-20
2-21/2-22	2-21/2-22
2-23/2-24	2-23/2-24
2-27/2-28	2-27/2-28
2-37/2-38	2-37/2-38
3-7/3-8	3-7/3-8
3-9/blank	3-9/3-10
5-1/5-2	5-1/5-2
	5-2.1/blank
6-1 through 6-11, blank	6-1 through 6-17, blank
A-1/A-2	A-1/A-2
A-3/A-4	A-3/A-4
A-5/A-6	A-5/A-6
A-7/A-8	A-7/A-8
A-9/A-10	A-9/A-10
	A-11/blank

File No.: 1S23

6/78
CB21A

MANUAL DIRECTORY

The following publications comprise the GCOS 6 manual set. The Manual Directory in the latest *GCOS 6 MOD 400 Systems Concepts* manual (Order No. CB20) lists the current revision number and addenda (if any) for each manual in the set.

Order No.	*Manual Title*
CB01	*GCOS 6 Program Preparation*
CB02	*GCOS 6 Commands*
CB03	*GCOS 6 Communications Processing*
CB04	*GCOS 6 Sort/Merge*
CB05	*GCOS 6 Data File Organizations and Formats*
CB06	*GCOS 6 System Messages*
CB07	*GCOS 6 Assembly Language Reference*
CB08	*GCOS 6 System Service Macro Calls*
CB09	*GCOS 6 RPG Reference*
CB10	*GCOS 6 Intermediate COBOL Reference*
CB20	*GCOS 6 MOD 400 System Concepts*
CB21	*GCOS 6 MOD 400 Program Execution and Checkout*
CB22	*GCOS 6 MOD 400 Programmer's Guide*
CB23	*GCOS 6 MOD 400 System Building*
CB24	*GCOS 6 MOD 400 Operator's Guide*
CB25	*GCOS 6 MOD 400 FORTRAN Reference*
CB26	*GCOS 6 MOD 400 Entry-Level COBOL Reference*
CB27	*GCOS 6 MOD 400 Programmer's Pocket Guide*
CB28	*GCOS 6 MOD 400 Master Index*
CB30	*Remote Batch Facility User's Guide*
CB31	*Data Entry Facility User's Guide*
CB32	*Data Entry Facility Operator's Quick Reference Guide*
CB33	*Level 6/Level 6 File Transmission Facility User's Guide*
CB34	*Level 6/Level 62 File Transmission Facility User's Guide*
CB35	*Level 6/Level 64 (Native) File Transmission Facility User's Guide*
CB36	*Level 6/Level 66 File Transmission Facility User's Guide*
CB37	*Level 6/Series 200/2000 File Transmission Facility User's Guide*
CB38	*Level 6/BSC 2780/3780 File Transmission Facility User's Guide*
CB39	*Level 6/Level 64 (Emulator) File Transmission Facility User's Guide*
CB40	*IBM 2780/3780 Workstation Facility User's Guide*
CB41	*HASP Workstation Facility User's Guide*
CB42	*Level 66 Host Resident Facility User's Guide*
CB43	*Terminal Concentration Facility User's Guide*

In addition, the following documents provide general hardware information:

Order No.	*Manual Title*
AS22	*Honeywell Level 6 Minicomputer Handbook*
AT04	*Level 6 System and Peripherals Operation Manual*
AT97	*MLCP Programmer's Reference Manual*
FQ41	*Writable Control Store User's Guide*

CONTENTS

ILLUSTRATIONS

TABLES

SECTION 2
LINKER

The Linker combines separately assembled and/or compiled object units, which can also be called compilation units (CUs), and produces a bound unit. An object unit can only be executed if it is first linked by the Linker. The Linker executes in either Short Address Form (SAF) mode (2 byte-address) or Long Address Form (LAF) mode (4 byte-address). It can create, in either mode, a SAF, LAF or SLIC bound unit. A SLIC bound unit can execute in either LAF or SAF mode.

Object units may contain external references to symbols.[1] While linking object units, the Linker resolves external references to symbols by referring to and updating a Linker-created symbol table. A link map of defined and/or undefined symbols can be produced.

To load the Linker into memory, enter the LINKER command (see "Loading the Linker" later in this section).

Linking is controlled by directives entered to the Linker through the directive input device. The directive input device is the device specified in the in_path argument of the "enter batch request" or "enter group request" command (normally, the in_path represents a terminal). This device can be reassigned in the command that loads the Linker.

If the Linker command specifies the –PT argument, the Linker prompter character "L?" will appear each time the Linker expects a directive.

The Linker processing can be interrupted by:

- Depressing the "QUIT", "INTERRUPT", or "BREAK" key on the user terminal
- Entering ΔCΔB group-id on the operator terminal, where group-id is the two-character group identification code associated with the group containing the task to be interrupted. A **BREAK** message appears on the user's terminal when the system interrupts the Linker. One of the commands SR (start), PI (program interrupt), UW (unwind) or NEW-PROC may be entered at this point. SR causes the interrupted task to resume at the point where the interrupt occurred (i.e., to continue as if no interrupt had occurred). If a MAP or MAPU directive has been issued and the PI command is used, the map operation is terminated at its current location and processing jumps to the next Linker directive. The UW command causes an orderly termination of the Linker processing (i.e., files are closed) and processing continues with some other task in the group containing the Linker.

The NEW-PROC command causes an orderly termination of the task group and the task group is reinitialized.

Each object unit to be processed during a single execution of the Linker must be a variable sequential file. The input files may reside in the same directory or in different directories. Unless specified otherwise, all of the object units are in the working directory (see "Specifying Location(s) of Object Unit(s) to be Linked" later in this section).

Only one bound unit is created by a single execution of the Linker. A bound unit may consist of only a root, or a root and one or more overlays. The root and each overlay may be up to 64K words (128K bytes). The root and each overlay is called a load unit; a load unit is loaded into memory by the Loader. When you use a create group or spawn group

[1] An external reference is a reference to a symbol defined in another object unit as an external symbol.

command, or an LDBU configuration directive, to request that a bound unit be loaded, the root is the portion of the bound unit that is loaded by the Loader. The root remains in memory as long as there are tasks executing on its behalf, unless LDBU was specified; if LDBU was specified, the root remains in memory until the system is reinitialized. An overlay is loaded into memory whenever it is required. Refer to the *Commands* manual for a discussion of the create group and spawn group commands, and the LDBU configuration directive.

Each bound unit has an attribute table associated with it; an attribute table contains information about the bound unit's characteristics and symbol definitions. The attribute table is loaded into memory immediately preceding the root.

SUFFIX CONVENTIONS

For input files, the Linker appends the suffix .O to each specified file name. When you specify a file name in a link directive, do not include a suffix. The Linker does not append a suffix to the output bound unit name specified in the system command line.

If a list file is designated (i.e., the -COUT argument is specified in the LINKER command), the Linker does not append a suffix to the specified name; otherwise, the Linker forms the name of its list file (Linker maps) by appending .M to the specified bound unit name.

Table 2-1 summarizes the formation of file names.

TABLE 2-1. DESIGNATING FILE NAMES

Program Preparation Task	Input File(s)	Output File(s)
Linker	Omit suffix. Linker appends .O to each specified file name.	Omit suffixes. The Linker appends .M to specified bound unit file name to form the name of the list file if the -COUT argument was not specified in the LINKER command. The Linker does not append a suffix to the name designated in the -COUT or -IN directives, nor to files named in the IN or LIB(x) directives.

FUNCTIONS OF THE LINKER

Creating a Bound Unit

The Linker produces a bound unit file whose pathname is specified in the name argument of the LINKER command.

The bound unit comprises only a root unless an OVLY or FLOVLY directive is entered. Each time an OVLY or FLOVLY directive is entered, the Linker initiates creation of a nonfloatable or floatable overlay, respectively. A nonfloatable overlay is loaded by the Loader into the same memory location (relative to the root) each time it is requested. A floatable overlay is linked at relative 0 (see "BASE Directive" later in this section), and can be loaded by the Loader into any available memory location. A floatable overlay must have the following characteristics:

1. External location definitions in the overlay are not referenced by the root or any other overlay.
2. There cannot be external references between floatable overlays.
3. The overlay does not contain external references that are not resolved by the Linker.
4. The overlay must be linked after all desired nonfloatable overlays have been linked.
5. The overlay cannot contain P+DSP references to any other overlay or the root.
6. The overlay cannot contain IMA (immediate memory address) references within itself.
7. There can be IMA references (with or without offsets) to locations in the root or any nonfloatable overlay.

NOTE: When the lowest address of a root or overlay has been established (i.e., an object unit has been linked), it is illegal to define a lower BASE address within that root or overlay.

START specifies the relative address at which the root or overlay will begin executing when it is loaded into memory by the Loader.

IST identifies the beginning of initialization code in the root.

SHARE designates that the bound unit is shareable.

SYS designates that the bound unit can be loaded into the system area as part of the system.

LINK, LINKN and LINKO specify which object units will be linked. The order in which specified object units are linked, and when they are linked, is determined by which link directive is specified.

OVLY names and assigns a number to the next nonfloatable overlay that follows, and designates the end of the preceding root or overlay.

FLOVLY names and assigns a number to the next floatable overlay that follows, and designates the end of the preceding root or overlay.

Call-cancel (CC) permits a COBOL program that used CALL and CANCEL statements to call overlays by their names.

QUIT designates that the last Linker directive has been entered. Execution of the Linker terminates after the bound unit has been created.

Producing Link Map(s)

Directives:
- MAP
- MAPU

A link map is written to the list file by specifying the MAP or MAPU directive. MAP creates a map that lists both defined and undefined symbols, whereas MAPU lists undefined symbols only.

Defining External Symbol(s)

Directives:
- COMM
- LDEF
- VAL
- VDEF
- EDEF

The COMM directive defines a symbol as being labelled or unlabelled common.[2]

A symbol can be defined as a relative location or value by specifying the LDEF or VDEF directive, respectively. The symbol's definition is then put into the symbol table by the Linker.

The VAL directive specifies a value definition at LINK time. This value is equivalent to the difference between two external locations.

The EDEF directive permits definitions in the Linker symbol table to be made part of the bound unit so they are available to the Loader at execution time.

[2] For discussions of "common" see the appropriate language reference manual.

Protecting or Purging Symbol(s)

Directives:
CPROT
CPURGE
PROT
PURGE
VPURGE

The CPROT and CPURGE directives, respectively, protect and remove symbols associated with labeled and unlabeled common.

The PROT and PURGE directives, respectively, protect and remove symbols and object unit names from the symbol table.

The protect (PROT) directive prevents certain symbols and/or object unit names from being removed from the symbol table. Symbols are protected if they identify a specified address or an address within a specified range; object unit names are protected if they are equated to a specified address or an address within a specified range.

The PURGE directive removes from the symbol table unprotected symbols that define a specified address or an address within a specified range, and/or object unit names equated to a specified address or an address within a specified range.

The VPURGE directive removes a specified value definition from the symbol table.

Designating That the Last Linker Directive Has Been Entered

Directive:
QUIT

QUIT must be the last Linker directive entered.

If a bound unit is being created, execution of the Linker terminates after the bound unit has been created.

If no bound unit is being created, QUIT terminates execution of the Linker.

LOADING THE LINKER

To load the Linker, enter the LINKER command, which is described below.

After the Linker is loaded, there is a typeout to the error output file of the revision also in the following format:

LINKER-nnnn-mm/dd/hhmm

where nnnn is a release identification, mm/dd is the month and day the Linker component was linked, and hhmm the time (hour, minutes) at which that link took place.

FORMAT:

LINKER bound-unit-path [ctl_arg]

ARGUMENT DESCRIPTIONS:

bound-unit-path
Pathname of the relative disk bound unit file. The pathname can be simple, relative, or absolute and must be preceded by a space. If the specified file already exists, the existing information in the file is deleted and replaced with the new bound unit. The bound unit pathname must be specified. It may be up to 62 characters in length.

ctl_arg

Control arguments; none or any number of the following control arguments may be entered, in any order:

-IN path

Pathname of the device through which Linker directives will be read; can be disk, card reader, operator's terminal, or another terminal.

Error messages are written to the error output file. Linker error messages are described in the *System Messages* manual.

Default: Device specified in the in_path argument of the "enter batch request" or "enter group request" command.

When this argument is specified, the prompter character will not appear.

-PT

If the -IN argument is not specified, -PT can be specified in order to produce a prompter character on the user terminal. A prompter character is issued only if -PT is specified.

-COUT list-path-name

Designates the list file. The list file can be sent to a disk, another terminal, or a printer. The list-path-name is associated with this list file. If -COUT is not specified, the list-path-name has a default value of bound-unit-path .M.

{-LAF / -SAF / -SLIC}

LAF and SAF are addressing modes in one of which the bound unit is to execute; -LAF designates long address form (two-word addresses); -SAF designates short address form (one-word addresses); -SLIC designates that either a SAF or a LAF machine may be used with no reassembly or link necessary.

Default: Bound unit executed in SAF (short address form) mode.

-SIZE nn

-SZ nn

nn designates the maximum number of 1024-word (1K) blocks of memory available for the Linker symbol table; nn must be from 1 to 32. At least 1024 words must be available.

Default: 2K

-W

Specifies that the implicit Linker work files are to be saved.

Default: Implicit Linker work files are automatically released by the Linker upon Linker termination..

-R

Designates that a bound unit is to be created, where all data areas defined as common are separated from all other code. Required for shareable CU's (object units).

-VERBOSE

Causes all Linker directives to be printed on the list file.

-NOMAP

Suppresses the list file.

Example:

```
LINKER MYPROG-INΔ MYDISK>CNLΔ-COUTΔ>SPD>LPT00Δ-SIZEΔ06
```

This LINKER command loads the Linker and designates the following:

- o Bound unit will be a relative file named MYPROG in the working directory.
- o Linker directives will be entered through disk file ^MYDISK>CNL.
- o List file goes to a line printer (configured as LPT00), rather than to a variable sequential file named MYPROG.M in the working directory.
- o The symbol table will be a maximum of 6K words of memory.

NOTE: LPT00 must have been previously defined in the DEVICE configuration directive, which is described in the "Startup and Configuration Procedures" section of the *System Building* manual.

ENTERING LINKER DIRECTIVES

Linker directives are entered through the directive input device, except for the following directives which may be embedded in assembly language CTRL statements: LINK, LINKN, LINKO, SHARE, EDEF, and SYS.

Linker directives comprise only a directive name or a directive name followed by one or more parameters. Each directive name *may be preceded by* 0, 1, or more blank spaces. If one or more parameters are to be specified in a Linker directive, the directive name *must be* immediately *followed by* one or more blank spaces.

Multiple directives can be entered on a line by specifying a semicolon(;) after each directive, except for the last directive on the line.

The last (or only) directive on a line can be followed by a comment; to include a comment, specify a space and a slash (/) after the last (or only) parameter and then enter the comment.

If the directive input device is the operator's terminal or another terminal, press RETURN at the end of each line (i.e., at the end of the comment, or at the end of the last directive if there is no comment).

If an error occurs when entering a directive, an error message is written to the error output file. Linker error messages are described in the *System Messages* manual. Determine what caused the error, and then reenter the directive correctly. If multiple directives are entered on a line and an error occurs, the error does not affect the execution of previously designated directives. The directive that caused the error and subsequent directives on that line are not executed.

PROCEDURE FOR CREATING ONLY A ROOT

To link object units and create only a root, load the Linker and then enter the following directives:

{LINK LINKN LINKO}[3]	Links object units.
QUIT	Designates that the last Linker directive has been entered. After the bound unit has been created, execution of the Linker terminates.

All other directives are optional.

PROCEDURE FOR CREATING A ROOT AND ONE OR MORE OVERLAYS

When creating a root and overlays, the following rules must be followed:

- o The root must be created before its overlays.
- o A root and all of its overlays must be created during the same execution of the Linker.

[3] Multiple LINK and/or LINKN and/or LINKO directives may be entered.

- o Nonfloatable overlays must be created before floatable overlays.
- o Overlays may contain references to symbols defined in the root or other overlays.
- o A root or overlay can be up to 64K words of memory.

To link object units and create a root and one or more overlays, load the Linker and then enter the following required directives:

{LINK / LINKN / LINKO}[4]	Links object units that will constitute the root.
{OVLY / FLOVLY}	Designates end of the root, and names and numbers the overlay that immediately follows.
{LINK / LINKN}	Links object units that will constitute an overlay.

NOTE: An OVLY or FLOVLY directive and at least one link directive must be specified for each overlay associated with the root.

QUIT	Designates that the last Linker directive has been entered. After the bound unit has been created, execution of the Linker terminates.

All other directives are optional.

NOTE: It is advisable to specify a MAP directive before each FLOVLY directive. The base address of a floatable overlay is relative 0, so all unprotected symbols that define locations will be purged from the symbol table.

PROCEDURE FOR CREATING A SHAREABLE BOUND UNIT USING A HIGH-LEVEL LANGUAGE

A shareable bound unit (BU) is one in which the code portion resides in system memory and can be used on behalf of one or more groups to manipulate data in that group. To accomplish this, the following factors must be present:

1. The pure (i.e., code) portion of the bound unit must be separated from the impure (i.e., data) portion.
2. The BU must be declared shareable.
3. Space must exist in the System pool to allow loading of the pure portion of the BU.

These factors are processed respectively as follows:

1. Using the capability to declare pure portions from impure portions (e.g., Intermediate COBOL), specify the -R argument on the Linker command line. This will cause the Linker to separate all those items declared as impure from the rest of the program.
2. Specify the SHARE directive for the BU at link time.
3. If both of the preceding conditions are specified, the Loader will automatically load the pure section of the BU into the System space in memory. If not enough room exists in the System space, the pure section will go into the group with the impure section and will no longer be shareable.

Using the Intermediate COBOL compiler, which automatically puts data in "Local Common", or using the Assembly Language pseudo-operator ($LOCOMW), the capability to share a pure code portion of a program exists. If the -R argument is specified at link time, the resultant BU can be up to 128K (up to 64K for pure code and up to 64K for data).

[4] Multiple LINK and/or LINKN and/or LINKO directives may be entered.

No overlays are permitted in a shareable/separated BU.

When the -R argument is specified, all data which the compiler defines in common is separated from executable code. All references in the code to this data are made via register $B6. The data does not directly reference the code.

When the -R argument is not specified, overlays are permitted. In this case, the maximum size of the root or of any individual overlay is 64K (including both code and data).

OBTAINING SUMMARY INFORMATION OF A LINKER SESSION

The Linker designates on the list file summary information regarding the bound unit created during the current execution of the Linker.

The list file includes the name of the bound unit and date and time of link, the name and revision number of each object unit linked, the name of the assembler/compiler, the assembler or compiler error count, and the sections described below:

ROOT	Name of the root.
HIGHEST OVLY	Number of the last overlay[5]; if there are no overlays HIGHEST OVLY is followed by a blank.
/NUM OF SYMS	Number of symbols specified in EDEF directives.
{SAF / LAF / SLIC}	Type of addressing form used in the bound unit; SAF is short-address form, and LAF is long-address form. A SLIC bound unit may be executed in either SAF or LAF mode.
{ROOT / OVLY}	Name of the root or overlay.
BASE	Base address of the root or overlay.
ST	Start address of the root or overlay.
SFUI	Specifies characteristics of the bound unit, as follows:

S
Shareable bound unit.

F
Floatable overlay(s) included.

U
There are resolved or unresolved forward references between the root and overlays or between overlays.

I
IMA addresses are present.

HIGH	Highest address in the root or overlay.
*SIZE OF ROOT AND STATIC OVLYS	Highest address in either the root or the largest overlay. (Indicates the amount of memory needed to load the bound unit.)
HI REL RCD	The number of the highest relative record of the bound unit file. (Indicates the number of control intervals used for storage.)

LINK DONE
Designates that execution of the Linker has been successful.

[5] The Linker assigns numbers to overlays. The first overlay is 00; subsequent overlays are numbered sequentially in ascending order.

The format for this information is illustrated below:

```
ROOT rootname
HIGHEST OVLY number/NUM OF SYMS number
**********
{SAF}
{LAF}
{SLIC}
**********
{CMMN[6]}rootname  BASE address ST address - . . . . HIGH = high address of data
{ROOT  }
{OVLY  } dirname {# overlay number} BASE address ST address - {S} {F} {U} {I} [7]
                 {Δ             }      {.} {.} {.} {.}
   HIGH = high address of root or overlay
**********
*SIZE OF ROOT AND STATIC OVLYS = number16   HI REL RCD = number10
**********
LINK DONE
**********
```

LINKER DIRECTIVE DESCRIPTIONS

Linker directives are described below, alphabetically. Some examples are provided to illustrate directive usage.

BASE Directive

The BASE directive defines, for subsequent object units to be linked, the relative link address within the bound unit. At load time, all addresses are relative to the beginning of available memory (relative 0) in the memory pool of the task group. When a task group is created, you specify the memory pool into which its bound units are to be loaded.

Unless BASE directives specify otherwise, the root will be linked, by default, at relative 0, and subsequent object units are linked at successive relative addresses. A BASE directive can be used at any point during linking to change the relative locations of the root, overlays, or individual object units. A floatable overlay always begins at relative 0; therefore, in a floatable overlay, BASE can be specified only *after* the first (or only) LINK, LINKN or LINKO directive. A BASE argument can specify a previously used or defined location, or an address relative to the beginning of the available memory.

If unprotected symbols define locations that are equal to or greater than the location designated in the BASE directive, those symbols are removed from the symbol table.

FORMAT:

```
BASE {$                                  }
     {%                                  }
     {X'address'                         }
     {=object-unit-name                  }
     {xdef [{+} X'offset']               }
     {      {-}                          }
     {# The current address.             }
```

[6] If –R argument is specified and common exists.
[7] This line is repeated for each overlay.

ARGUMENT DESCRIPTION

$
Next location after the highest address of the linked root or previously linked nonfloatable overlay.

% absolute
Highest address+1 ever used in the linked root or *any* previously linked nonfloatable overlay.

address
Hexadecimal address comprising one to four integers enclosed in apostrophes and preceded by X. The specified address is relative to the beginning of available memory (relative 0) in the memory pool at load time.

=object-unit-name
Specified object unit's base address; the subsequent root, overlay, or object unit will be linked at the same relative address as the specified object unit, which must have already been linked. Furthermore, the object unit name must still exist in the symbol table (i.e., it is not purged).

xdef$\left[\left\{\begin{matrix}+\\-\end{matrix}\right\}\text{X'offset'}\right]$
Address of any previously defined external symbol. If an offset is specified, it must be a hexadecimal integer with an absolute value less than 8000 (32768 decimal).

Default:
Root–0
Nonfloatable overlay-Next location after the highest address of the preceding root or nonfloatable overlay
Floatable overlay–0

Example:

This example illustrates usage of BASE directives in a bound unit that comprises a root and overlays. In this example, assume that the bound unit being created is going to be executed as part of task group A1, and memory pool AA is to be used by this task group. Figure 2-1 illustrates memory pool AA's location in memory relative to the system pool and another pool, and the locations within that memory pool to which each object unit specified in the following directives will be loaded.

LINKER TEXTΔ-COUTΔ>SPD>LPT00 START TEXTEN	Designates address at which execution will begin when the root is loaded.
IST INIT	Defines INIT as the beginning of initialization code.
LINK OBJ1,OBJ2	Request that OBJ1.O and OBJ2.O be linked.
MAP	Causes OBJ1.O and OBJ2.O to be linked, and produces a link map.
OVLY ABLE	Designates end of the root, and that a nonfloatable overlay named ABLE immediately follows. The Linker assigns the number 00 to this overlay.

CC (Call-Cancel) Directive

The call-cancel directive (CC) must be used when linking COBOL programs that contain CALL/CANCEL statements that reference overlays. The Linker will place each overlay name and its associated Linker-generated overlay number into the bound unit attribute table so that the COBOL program can call/cancel overlays by name.

To support the CALL/CANCEL facility, the object unit ZCCEC is required. ZCCEC will be automatically linked into the root; it requires no link directive.

The CC directive must be specified before the first LINK, LINKN or LINKO directive in the root.

FORMAT:

CC

COMM Directive

The COMM directive defines a labelled or unlabelled "common" area of a specified size.

FORMAT:

COMM symbol, size

ARGUMENT DESCRIPTION:

symbol
Identifies the external symbol which is to be treated as common.

size
Size is specified as a 1- to 4-character hexadecimal number bound by single quotes and preceded by the letter X (i.e., X'size').

CPROT Directive

The CPROT directive prevents specified symbols from being removed from the common area.

FORMAT:

CPROT symbol

ARGUMENT DESCRIPTION:

symbol
Name of the external symbol, that is to be protected. The symbol must be specified in the COMM directive, or defined as common at the time of assembly or compilation.

CPURGE Directive

The CPURGE directive causes the Linker to remove an unprotected symbol from the common area.

FORMAT:

CPURGE symbol

ARGUMENT DESCRIPTION:

symbol
Identifies the external symbol which is to be removed from the common area.

EDEF Directive

The EDEF directive causes the transfer of a symbolic definition from the Linker to the Loader at load time. The bound unit attribute table is part of the bound unit.

An EDEF directive can only specify a symbol that has been defined using XDEF, LDEF, or VDEF. When EDEF is specified, the symbol's definition must already be in the symbol table.

Secondary entry points of bound units, whose code is to execute under control of a task, must be defined in an EDEF directive. This includes secondary entry points of overlays and the root entry point when it will be explicitly used in a create group command. The start address of the root and of each overlay is placed by the Linker in the bound unit attribute table and does not need an EDEF definition.

If a bound unit is memory resident, symbols (entry points and references) can be defined by EDEF so that they can be referenced by any bound unit loaded by the system. At system configuration time, when the resident bound units are loaded using the LDBU system configuration directive, these symbols are placed in the system symbol table. When the Loader loads other bound units that contain unresolved references, it tries to resolve them with the list of symbols defined for resident bound units.

If the bound unit is transient (shareable or not shareable), the symbols in the attribute table of the bound unit are meaningful only as definitions of secondary entry points. Although shared bound units can be in the address space of more than one task group, the bound unit attribute table is available to the Loader only when the bound unit is being loaded. Unresolved references in any bound unit will be resolved only to symbols defined in attribute tables of resident bound units.

The EDEF directive can be embedded in assembly language CTRL statements.

FORMAT:

$$\left\{\begin{matrix}\text{EDEF}\\ \text{EF}\end{matrix}\right\}\text{symbol}$$

ARGUMENT DESCRIPTION:

symbol
Any external definition comprising one to six characters. The symbol must have been defined. If the symbol was multiply defined, the first definition is used.

Example:

This example illustrates usage of EDEF directives in bound units.

LINKER MYPROG	Loads the Linker. The bound unit named MYPROG will be created on the working directory. The list file MYPROG.M is also created on the working directory.
LINK A	
LINKN B	
MAP	
EDEF B	B is a symbol defined as an external location or value in B.O.
LDEF SYM,X'1234'	Assigns relative location 1234 to external symbol named SYM.
OVLY FIRST	Designates end of root, and names nonfloatable overlay that immediately follows.

LINK X,Y EDEF SYM QUIT	Designates that the last Linker directive has been entered. Execution of the Linker terminates after the bound unit has been created.
LINKER PROG2 -COUT >SPD> LPT00 -SIZE 02	Loads the Linker; the bound unit to be created is named PROG2. The list file is the printer. The symbol table is a maximum of 2K words of memory.
BASE X'2222'	Subsequent object units will be loaded into memory starting at the relative address 2222.
LINKN W	Requests that object unit W.O be linked.
MAP	Produces a link map; in this map, it is determined that object unit W.O contains an unresolved reference to the symbol SYM, which was defined in the root of the bound unit MYPROG.
QUIT	

If MYPROG is loaded into memory via an LDBU configuration directive, when the Loader loads PROG2 the Loader will resolve the unresolved reference in PROG2 to the symbol SYM, which was defined in the root of MYPROG.

NOTE: An EDEF directive cannot be entered on the directive line in which the object unit is specified. For example, if the symbol TAG is defined in object unit A, the following directive line is *not* allowed:

LINK A;EDEF TAG

FLOVLY Directive

The FLOVLY directive assigns the specified name and a number to the *floatable* overlay that immediately follows, and designates the end of the preceding root or overlay. The characteristics of floatable overlays are described earlier in this section under "Creating a Bound Unit."

FLOVLY must be specified as the first directive of each floatable overlay. Floatable overlays must be linked after all desired nonfloatable overlays have been linked.

The Linker assigns a two-digit number to each overlay. Overlays are numbered sequentially, in ascending order; the first overlay is 00.

FORMAT:

FLOVLY name

ARGUMENT DESCRIPTION:

name

Name of the floatable overlay that immediately follows. The overlay name must comprise one to six alphanumeric characters; the first character must be alphabetic.

Example:

LINKER BU	Loads the Linker and designates BU as the bound unit name.
LINK A LINK B MAP	Produces a link map. The link map should be referenced to determine if there are any unprotected symbols that define locations. These symbols, if any, will be removed from the symbol table since the floatable overlay that immediately follows has a default base address of 0.
FLOVLY GR	Designates the end of the root (which comprises object units A.O and B.O), and specifies that the next overlay is a floatable overlay named GR. The Linker assigns the number 00 to this overlay.
LINK X LINK Y MAP FLOVLY BR	Designates the end of floatable overlay GR, and designates that the floatable overlay that immediately follows is named BR. The Linker assigns the number 01 to this overlay.
LINK R6 MAP QUIT	

IN Directive

The IN directive designates a different directory as the primary directory.[8] This directive permits the linking of object units that are in directories other than the default primary directory or secondary directory (if any). If the IN directive is not specified, the working directory is the primary directory. (The secondary directory is designated in the LIB directive.)

NOTE: The IN directive must be specified before the first LINK, LINKN or LINKO directive that requests the linking of an object unit that is in the specified directory.

The specified directory remains the primary directory until another IN directive is entered. If the primary directory is changed via an IN directive and at a later time you want the task group's working directory to be the primary directory, you may enter the IN directive and omit a pathname.

FORMAT:

IN [path]

[8] The primary directory is the first directory that the Linker searches for the specified object unit(s) to be linked.

ARGUMENT DESCRIPTION:

[path]
Pathname of the directory being designated as the primary directory. The pathname may comprise a maximum of 64 characters. A simple, relative, or absolute pathname may be specified (methods of designating pathnames are described in the *Program Preparation* manual). If path is omitted, the working directory becomes the primary directory. This argument may not be embedded in source code (CTRL).

Example 1:

```
INΔ^DIR>PRIM
```

This directive designates that ^DIR>PRIM is the primary directory.

Example 2:

This example illustrates usage of the IN directive in conjunction with directives that request the linking of object units. Assume the primary directory is the working directory, whose relative pathname is WORK>CURR; object units X.O, Y.O, and Z.O are in the working directory.

LINKER OUTPUT	Loads the Linker; a bound unit named OUTPUT will be created on the working directory.
LINK X	Requests the linking of object unit X.O; X.O is in the working directory.
INΔ^NEW>PRIM	Designates that ^NEW>PRIM is now the primary directory.
LINK A	Requests the linking of object unit A.O, which is in the primary directory. ^NEW>PRIM>A.O is the pathname of A.O, as expanded by the Linker.
LINK C	Requests the linking of object unit C.O, which is in the primary directory. ^NEW>PRIM>C.O is the pathname of C.O, as expanded by the Linker.
IN WORK>CURR	Designates that the primary directory is now the working directory.
LINKN Y	Requests the linking of object unit Y.O, which is in the working directory. WORK>CURR>Y.O is the pathname of Y.O, as expanded by the Linker.
MAP	
QUIT	

IST Directive

The IST directive identifies the beginning of initialization code in the root. Initialization code is to be executed once only, immediately after the root is loaded. After the initialization code is loaded, the space may be made available for overlays. The IST directive is meaningful only when associated with an LDBU directive that specifies an initialization subroutine table (IST). LDBU, a CLM directive, is explained in the *System Building* manual.

FORMAT:

$$\left\{\begin{matrix}\text{IST}\\ \text{IT}\end{matrix}\right\} \text{external symbol}$$

ARGUMENT DESCRIPTION:

external symbol
Symbol specified by label in IST section of LDBU.

LDEF Directive

LDEF assigns a relative *location* to an external symbol. A symbol should be defined only once, either as a location or as a value. When a symbol is defined, its definition is put into the Linker symbol table so that it can be used to resolve references to the symbol during linking. When a symbol defined as a location is no longer referenced, its symbol table entry can be cleared by specifying the PURGE directive. PURGE has no effect if a protect (PROT) directive was previously specified.

FORMAT:

$$\left\{\begin{matrix}\text{LDEF}\\\text{LF}\end{matrix}\right\}\ \text{symbol,}\ \left\{\begin{matrix}\$\\\%\\\text{X'address'}\\\text{=object-unit-name}\\\text{xdef}\left[\left\{\begin{matrix}+\\-\end{matrix}\right\}\ \text{X'offset'}\right]\\\#\end{matrix}\right\}$$

ARGUMENT DESCRIPTIONS:

symbol
One to six alphanumeric characters.

$
Next location after the highest address of the linked root or *previously* linked nonfloatable overlay.

%
Highest address+1 ever used in the linked root or *any* previously linked nonfloatable overlay.

address
Hexadecimal address comprising one to four integers enclosed in apostrophes and preceded by X. The specified address is relative to the beginning of available memory (relative 0) in the memory pool.

=object-unit-name
Specified object unit's base address.

xdef[{+ -} X'offset']
Address of any previously defined external symbol. If an offset is specified, it must be a hexadecimal integer with an absolute value less than 8000 (32768 decimal).

#
The current address.

Example:

This example illustrates usage of each format of the LDEF directive.

LINKER BOUND	Loads the Linker and designates BOUND as the bound unit name.
LINK A	
LINK B,C	
MAP	
LDEF SYM,X'1234,	SYM assigned relative location 1234
OVLY FIRST	Designates end of root and names first nonfloatable overlay

```
LINK R
MAP
LDEF QUIZ,=C
```

QUIZ assigned base location of the previously linked object unit named C.O.

```
OVLY SECOND
LINKN D
LINK F
MAP
LDEF NEW,SYM
```

NEW assigned same location as the symbol SYM, which was defined in the root; i.e., NEW is assigned relative location 1234.

```
OVLY NEXT
BASE X'1300'
LINK W,X
MAP
LDEF ANY,$
```

ANY assigned next location after highest address of the previously linked nonfloatable overlay, SECOND.

```
OVLY THIRD
LINK Z
LINK Q
MAP
LDEF FIND,%
```

FIND assigned next location after highest address of the root or *any* previously linked nonfloatable overlay. (A previous nonfloatable overlay was named SECOND; if it ended at location 1566 and this is the highest address ever reached during the linking of object units constituting this bound unit, FIND would be assigned location 1567.)

```
QUIT
```

LIB Directive

The LIB directive designates a directory as the secondary directory. This directory permits the linking of object units that are in a directory other than the primary directory. If an object unit specified in the LINK, LINKN or LINKO directive cannot be found in the primary directory, the Linker then searches the secondary directory.

If LIB is not specified, there is no secondary directory; the Linker searches only the primary directory.

The specified secondary directory remains in effect until the LIB directive is respecified with a different directory name, or without any directory name.

NOTE: The LIB directive must be specified before the first LINK, LINKN or LINKO directive that requests the linking of an object unit that is in the secondary directory. This directive may not be embedded in source code (CTRL).

FORMAT:

LIB [path]

ARGUMENT DESCRIPTION:

[path]
Pathname of the directory being designated as the secondary directory. A simple, relative, or absolute pathname may be specified. (Methods of specifying pathnames are described in Section 1.) If path is omitted, no search of a secondary directory is made.

LIB/LIB(2, 3, or 4)
Example 1:

```
LIB DIR>SECND
```

This directive designates that DIR>SECND is the relative pathname of the secondary directory.

Example 2:

This example illustrates usage of a secondary directory that contains object units W.O, Y.O, and Z.O.

LIB DIR>SECND	Designates that DIR>SECND is the relative pathname of the secondary directory.
LINK B	Requests the linking of object unit B.O; B.O resides in the primary directory.
LINK A	Requests the linking of object unit A.O; A.O resides in the primary directory.
LINK W	Requests the linking of object unit W.O; W.O resides in the secondary directory. DIR>SECND>W.O is the full pathname of W.O, as expanded by the Linker.

All specified object units in the primary directory are linked first; then all specified object units in the secondary directory are linked, and so on. To cause object units to be linked in a specific order, the LINKN or LINKO directive must be used.

LIB $\begin{Bmatrix}2\\3\\4\end{Bmatrix}$ Directive

The LIB (2, 3, or 4) directive designates directories as the third, fourth or fifth directory. If an object unit specified in the Linker directive cannot be found in the primary or secondary directory, then the third directory is searched and so on.

The specified directories remain in effect until another LIB (2, 3 or 4) statement is given.

NOTE: The LIB (2, 3 or 4) directive must be specified before the first LINK, LINKN or LINKO directive that requests the linking of an object unit that is in one of these directories.

FORMAT:

$$\text{LIB} \begin{Bmatrix}2\\3\\4\end{Bmatrix} [\Delta\text{path}]$$

ARGUMENT DESCRIPTION:

path
Pathname of the third, fourth or fifth directory to be searched (if LIB is specified) if the object unit specified in a Linker directive is not found in the preceding directories. A simple, relative or absolute pathname may be specified. If path is omitted, the specified directory (2, 3, or 4) is removed from the list of directories to be searched by the Linker.

LINK Directive

The LINK directive specifies that the Linker link one or more specified object units. Each specified object unit name is put into the link request list. The object units are linked when the first subsequent directive other than LINK or START is encountered. When this occurs, the Linker searches the primary directory and links the specified object units in the order in which they were requested. If all of the object units are not found and there is a secondary directory, the Linker searches the secondary directory and links specified object units, in the order in which they were requested. If there is a copy of an object unit in both the primary and secondary directory, the copy in the primary directory is linked.

The order in which object units are linked is important for the following reasons: (1) it determines which object units will be in memory simultaneously and which object units will overlay other object units and (2) within the root and each overlay, the first start address encountered by the Linker (either in an END statement or a START directive) is used as the start address for that root or overlay.

During each execution of the Linker, at least one LINK, LINKN or LINKO directive must be entered for each root or overlay. Multiple LINK directives can be specified within a single root or overlay. If LINK and/or LINKN and/or LINKO directives request that the same object unit be linked more than once within a single bound unit, only the first request is honored.

LINK directives can be embedded in assembly language CTRL statements; the specified object unit(s) are added to the link request list immediately following the object unit in which they were embedded. See "LINKN Directive" and "LINKO Directive" for the order in which object units are linked if there are embedded LINK directives and/or LINKN and/or LINKO directives.

FORMAT:

LINK obj-unit$_1$ [,obj-unit$_2$]...

ARGUMENT DESCRIPTION:

object-unit$_n$
: Name of an object unit to be linked. An object unit name consists of one to six characters, each of which must be an alphanumeric character or a dollar sign ($), a period (.), or an underbar (_). If multiple object units are specified, they are linked in the order convenient to the Linker. The first character must be a letter or dollar sign ($).

Example 1:

```
LINK FIRST
```

This directive causes the Linker to link the object unit named FIRST.O. The primary directory is searched first; if FIRST.O is not found, the secondary directory, if any, is searched.

Example 2:

```
LIB SECOND>FILE
LINK R
LINK T
```

The above LIB directive designates that SECOND>FILE is the pathname of the secondary directory. In this example, object unit R.O is in the secondary directory, and object unit T.O is in the primary directory.

The above LINK directives will link T.O before R.O, since T.O is in the primary directory.

Example 3:

```
LINK A,B,C,D
```

This directive causes the Linker to link the object units named A.O, B.O, C.O, and D.O. If the primary directory contains B.O, and the secondary directory contains A.O, C.O, and D.O, the object units are linked in the following order:

B.O
A.O
C.O
D.O

LINKN Directive

The LINKN directive causes object units to be linked in the following order:

1. Object units previously specified in LINK directives, and any object units requested in embedded LINK directives. The object units are linked in the order in which they are found by the Linker.
2. First (or only) object unit specified in the LINKN directive.
3. Object units specified in LINK and/or LINKN directives that are embedded in the object unit linked as a result of step 2 above.
4. Additional object units, if any, specified in the LINKN directive; the object units are linked in the order in which they were specified in LINKN, regardless of whether they are in the primary or secondary directory. If an object unit contains an embedded directive to link another object unit, the object unit designated in the embedded directive is linked after the object unit that contains the embedded directive.

If directives designate that an object unit be linked more than once within a single bound unit, only the first request is honored, unless intervening directives are specified that result in the first linked object unit being overlays with other code at execution time.

During each execution of the Linker, at least one LINKN, LINK or LINKO directive must be specified for each root or overlay.

Multiple LINKN directives can be specified within a single root or overlay.

LINKN directives can be embedded in assembly language CTRL statements; the specified object unit(s) are added to the link request list immediately following the object unit in which they were embedded.

FORMAT:

$$\left\{\begin{matrix}\text{LINKN}\\\text{LN}\end{matrix}\right\} \quad \text{obj-unit}_1 \ [,\text{obj-unit}_2\,]\ldots$$

ARGUMENT DESCRIPTION:

obj-unit$_n$
: Name of an object unit to be linked. An object unit name must be one to six alphanumeric characters and must not include a suffix; the first character must be a letter or dollar sign ($). The Linker appends the suffix .O to each object unit name, and searches for the specified object unit name, including the suffix.

Example 1:

```
LINKN X,W
```

This directive designates that the Linker link the object unit named X.O and then link the object unit named W.O.

If there are external references in both P-relative and immediate memory address forms to an undefined symbol, the symbol is listed twice under UNDEF.

Figure 2-2 illustrates the formats of maps generated by the MAP and MAPU directives. In a single-word (SAF) system, each address or value is specified in four hexadecimal digits; in a double-word (LAF) system, each address or value is specified in eight hexadecimal digits.

NOTE: The date and time at which the bound unit was created is automatically put in the bound unit's attribute section.

```
 * *   bound unit name  LINK MAP  yyyy/mm/dd  hhmm:ss.s
 * * START    address
 * * LOW      address
 * * HIGH     address
[**$COMM address]
 * * CURRENT address
 * * EXT DEFS
 P     ZHCOMM[a]            0000 [0000]
 P     ZHREL[a]             0000 [0000]
 * *   ROOT                 base address of root
[P]*   object unit name     base address of object unit
[P][M/C] symbol name[b]     address[c] or value
         .                    .
         .                    .
         .                    .
[P]*   object unit name     base address of object unit
[P][M/C] symbol name[b]     address[c] or value
         .                    .
         .                    .
         .                    .
 * *   overlay name         base address of overlay
[P]*   object unit name     base address of object unit
[P][M/C] symbol name[b]     address[c] or value
         .                    .
         .                    .
         .                    .
[P]*   object unit name     base address of object unit
[P][M/C] symbol name[b]     address[c] or value
         .                    .
         .                    .
         .                    .
[* * COMMON                  common definitions are separated on the map as well as in the bound
                             unit when -R is specified]
 * * UNDEF
```

OMITTED IF MAPU SPECIFIED

Figure 2-2. Link Map Formats

```
[P]*   object unit name[d]   base address of object unit
       [symbol name[b]       address of most recent reference[e]]
          .                     .
          .                     .
          .                     .
[P]*   object unit name[d]   base address of object unit
       [symbol name[b]       address of most recent reference[e]]
          .                     .
          .                     .
          .                     .
```

P - Protected symbol

M - Multiply defined symbol

C - Symbol defines labeled or unlabeled common

[a] ZHCOMM and ZHREL are reserved symbol names; they appear on every map as protected symbols. ZHCOMM is located at unrelocatable zero. ZHREL is located at relocatable zero. When ZHCOMM is used in an LDEF directive, the new symbol will not have the attribute of being non-relocatable.

[b] The map contains the names of all external symbols currently defined in the symbol table. If there are external references in both P-relative and immediate memory address forms to an undefined symbol, the symbol is listed twice under UNDEF. Each map line contains up to four (SAF) or three (LAF) external symbols.

[c] To find a location definition, add the relocation factor at load time to the address shown on the map.

[d] All objects units linked are listed under UNDEF, even if they contain no unresolved references.

[e] Within the root or a single overlay, the latest reference to an undefined symbol need not be in the object unit that contained the first reference to the symbol. For each undefined symbol, the following information is given under UNDEF: name of the first object unit that contains a reference to the designated symbol, and the relative address of the most recent reference.

Figure 2-2 (cont.) Link Map Formats

Figure 2-3 presents sample link maps.

OVLY Directive

The OVLY directive assigns the specified name and a number of the *nonfloatable* overlay that immediately follows, and designates the end of the preceding root or overlay.

OVLY must be specified as the first directive of each nonfloatable overlay.

The Linker assigns a two-digit *number* to each overlay. Overlays are numbered sequentially, in ascending order; the first overlay is 00.

FORMAT:

OVLY name

ARGUMENT DESCRIPTION:

value-definition
The external symbol associated with a particular value.

EXAMPLE ILLUSTRATING USAGE OF THE LINKER

LINKER TEST -COUT >SPD>LPT00	The bound unit will be a relative file named TEST created in the working directory. Link maps will be printed on the printer configured as LPT00.
START LOC	
IST INITST	Defines the beginning of initialization code.
LINK OBJ1	Requests that OBJ1.O be linked.
LIB ^DSK03	Names secondary directory.
LINK OBJ2	Requests that OBJ2.O be linked.
OVLY ABLE	Causes OBJ1.O and OBJ2.O to be linked, designates the end of the root, and specifies that a nonfloatable overlay named ABLE immediately follows. The Linker assigns the number 00 to this overlay.
LINKN OBJ3	
LINKN OBJ4	
PROT =OBJ3	Protects the symbol OBJ3. This symbol is protected because a subsequent overlay may be loaded starting at the base address of OBJ3.O.
MAP	Requests a link map.
OVLY BAKER	Designates the beginning of the nonfloatable overlay named BAKER. The Linker assigns the number 01 to this overlay.
LINKN OBJ5	
LINKN OBJ6	
PROT =OBJ5	Protects the symbol OBJ5.
MAP	
OVLY DOG	Designates the beginning of the nonfloatable overlay named DOG. The Linker assigns the number 02 to this overlay.
BASE =OBJ5	The overlay named DOG will be loaded starting at the address where overlay BAKER began.
LINK OBJ7	
MAP	
OVLY FOX	Designates the beginning of the nonfloatable overlay named FOX. The Linker assigns the number 03 to this overlay.
BASE =OBJ3	FOX will be loaded at starting address of overlay ABLE.
IN ^DSK01>MYFILE	Designates that the primary directory now is the directory named ^DSK01>MYFILE.

```
VPURGE
LIB ^DSK02>MYLIB          Designates that the new secondary directory
                          is named ^DSK02>MYLIB; if necessary, this
                          directory will be searched after the primary
                          directory.
LINK OBJA
LINK OBJB
MAP
OVLY X-RAY                A nonfloatable overlay named X-RAY
                          immediately follows. The Linker assigns the
                          number 04 to this overlay.
BASEΔ=OBJ5                X-RAY will be loaded starting at the
                          beginning address of BAKER.
LINK OBJC
MAP
FLOVLY FLOAT              Designates that a floatable overlay named
                          FLOAT immediately follows. The Linker
                          assigns the number 05 to this overlay.
LINK OBJE
MAP
QUIT
```

PROGRAMMING CONSIDERATIONS

1. While processing object units, the Linker creates a work file LNKWRK.W in the working directory. This file is a variable sequential file. It is initially allocated with four control intervals of 256 bytes each, but it can be expanded to the amount of space available in the working directory. If the bound unit is large, link execution time may be reduced by the use of preallocated files.
2. If the relative output file is preallocated, it must have the same name as that specified in the name argument of the LINKER command, it must be a fixed, relative file, and it must have a record size of 256 bytes.
3. If multiple object units contain labeled and unlabeled common, the object units will be linked with common blocks appearing in the following order (-R is not specified):
   a. Labeled or unlabeled common (defined in first object unit linked)
   b. First object unit (including external references and definitions)
   c. Labeled common (defined in second object unit linked)
   d. Second object unit (including external references and definitions)
   e. Object unit n
4. A root or any overlay may reference any symbol defined in any other root or overlay including "common" symbol definitions. A common area cannot, however, be initialized in any overlay other than the one in which it initially occurs (is made known to the Linker). That is, a common area defined in a root or an overlay can be initialized only in the root or overlay in which it is defined.
5. Relocation can occur during one or both of the following procedures:
   a. Assembly; by specifying an ORG statement, subsequent object text within the object unit is relocated. (See the *Assembly Language Reference* manual.)
   b. Linking; by specifying the BASE directive, subsequent object units to be linked within the root or overlay have a specified relative load address. (See "BASE Directive" earlier in this section.)

-POOL id

id is a two-character ASCII identifier and is the name of the memory pool from which all memory required by the spawned task group is to be taken. If specified, id must have been defined at system building time. If not, the issuing task group's memory pool is used.

-ARG arg arg . . . arg

Indicates that additional arguments required by the spawned task group during execution follow. These additional arguments are passed to the lead task of the spawned group to be used as necessary, and are substituted for parameters in the command-in file. If used, the -ARG control argument must appear last. Refer to the *Commands* manual for an explanation of the use of additional arguments.

NOTE: In any invocation of the SG command, -EFN *or* ECL, but not both, can be specified. If neither is specified, -ECL is assumed and the in_path argument is required.

Using the Login Command

The login command is used to gain access to the system. The login command is entered from any terminal not designated as a direct-login terminal or an abbreviated-login terminal. (To determine the type of terminal he is at, the user should contact the installation supervisor.) The login command causes a task group associated with the user's terminal to be spawned. Once he has access to the system, the user cannot again invoke login unless he first issues the BYE command or the task group is otherwise terminated.

FORMAT

L [login id] [destination id] [etl_arg]

ARGUMENT DESCRIPTION:

login id

Establishes the identity of the user who is attempting to gain access to the system. Provides the user identification for the spawned task group. The login_id argument consists of from one to three fields having the following meanings:

person

person.account

person.account.mode

person	Name of person who may access system; can be from 1 through 12 characters. (For example, WDSMITH could be the value for the person field.)
account	Name of an account under which the user is to work; can be from 1 through 12 characters. (For example, JSINVENTORY could be used as the value for the account field.)
mode	Provides a further identification of the user; can be from 1 through 3 characters. (For example, VER could be used as the value for this field.)

[destination_id]

Optional argument that permits the user to login as a secondary user of an existing task group. (It is necessary that the running task group have previously issued a request for a secondary user terminal; the request for a secondary user terminal is entered by means of a Request Terminal macro vall; see the *GCOS6 System Service Macro Calls* manual for the format of the Request Terminal macro call.) To login as a secondary user of a user-created applications program, the user enters the value nn, where nn is the task group id of the task group in which the application is running. To login as a secondary user of task group $T (Terminal Concentrator), see the *Terminal Concentration Facility User's Guide.* When destination–id is specified, no control arguments can be selected. If the secondary login capability is not desired, then destination_id is omitted.

ctl_arg

One or more of the following control arguments can be selected:

{-PO path [id]
{-PO * id }

Used to override the default lead task and group id/pool id specifications for the task group spawned as a result of this login procedure.

path

Pathname of the bound unit to be executed as the lead task of the spawned task group. If this argument is omitted, the lead task is the command processor.

id

Group id/pool id of the spawned task. The group id and the pool id are represented by the same 2-character value. If this argument is not specified, the group id is a 2-character value whose left (first) character was specified when the Listner component was activated (the Listner is described in the *Operator's Guide*) and whose right (second) character is the next unused character in the sequence 0 through 9 and A through Z, as selected by the system.

-HD path
Used to override the default working directory specification for the task group spawned as a result of the login procedure.

path
Pathname of the working directory for the spawned task group. If this argument is omitted, the working directory pathname is null.

-LRN n
Used to override the default maximum logical resource number (LRN) value for the task group spawned as a result of this login procedure.

n
Maximum LRN value to be used for the spawned task group. (The maximum possible LRN value is 252.) If this argument is omitted, the maximum LRN value is the highest value in the system group.

-LFN n
Used to override the default logical file number (LFN) value for the task group spawned as a result of the login procedure.

n
Maximum LFN value to be used for the spawned task group. (The maximum possible LFN value is 255.) If this argument is omitted, the maximum LFN value is 15.

-ARG arg arg . . . arg
Passes additional arguments to the task group spawned as a result of this login procedure. These additional arguments are passed to the spawned task in an extension of the task request block, and are substituted for parameters in the command input file. If used, the -ARG control arguments must appear last. Refer to the *Commands* manual for an explanation of the use of the additional arguments.

The arguments will be substituted in the following manner:

- Argument 1 will always be null
- If the lead task is the command processor, argument 2 will be the pathname of the user-in file (i.e., >SPD>terminal) and arguments 3 through n will be the arguments following -ARG.
- If the lead task is not the command processor, arguments 2 through n will be those arguments following -ARG.

SECTION 5
DEBUGGING PROGRAMS

While a program is executing, it can be monitored by using Debug. If there is not enough room in memory for Debug, you can monitor a program by temporarily leaving space in the program or by using Patch to append monitor points. (See "Debugging Programs Without Using Debug" later in this section.)

DEBUG

Debug provides patching and testing facilities for application programs running under the operating system. Debug runs as its own task group.

Program testing and error correction is performed as an interactive dialogue between the operator and Debug. Execution of Debug is controlled by directives entered to Debug. Addresses used with Debug are system-wide absolute memory addresses; therefore, Debug directives are effective across task and task group boundaries. Debug directives are entered through the device specified as user-in in the request to establish the Debug task group (i.e., a user-specified terminal).

The following functions can be performed using Debug:

- Define, store, and execute a sequence of directives either entered through the input device, or referenced when a breakpoint directive or trace trap (BRK generic instruction) is encountered in the load unit being tested.[1]
- Set or clear breakpoints in task code to monitor task status. (Breakpoints are described in detail later in this section.)
- Set or clear breakpoints in bound units, to gain control of bound units as they are loaded.
- Display, change, and dump either memory or registers; information may be printed on a line printer, the operator terminal, or another terminal.
- Evaluate expressions.

Debug File Requirements

Debug directives stored for later execution reside in a preallocated, relative disk file DEBUG.WORK (these directives are identified and described in Table 5-2, "Summary of Debug Directives, by Function," later in this section). The file DEBUG.WORK must be in the volume major directory of the disk device referenced in the specify file (SF) directive. (The SF directive is described later in this section.)

Loading the Debug Task Group

Debug requires a minimum memory area or pool of 2000 words in which to execute. Use the MEMPOOL directive during initialization to create such a memory pool and to specify the pool's identification (see the *Commands* manual for details about MEMPOOL).

Example:

```
MEMPOOL ,AB,2000
```

This MEMPOOL directive creates a nonexclusive memory pool comprising 2000 words that can be specified when the Debug task group is loaded into memory.

[1] Breakpoints and trace traps either cause a specified Debug directive line to be executed, or interrupt execution of the task so that its status can be determined.

Debug is loaded into the system as the lead task of a dedicated task group named $D. The base level number of the Debug task group is treated as a physical level instead of a value relative to the configured system, so that Debug may have priority over system tasks. The Debug task group must be assigned two priority levels which are not assigned to other tasks or task groups.

The following examples illustrate methods of loading Debug. Example 1 illustrates a spawn group command. Example 2 illustrates a create group request and an enter group request. The following description applies to both examples:

The Debug task group's identification is $D, your identification is GALE.TECH, and the base priority level of Debug is 7. Debug will use levels 7 and 8. Directives to Debug will be entered through the operator terminal, which is identified by its pathname >SPD>CONSOLE. The bound unit DEBUGDB will be loaded, if necessary, and execute as the task group's lead task.

Example 1:

Loading Debug by a spawn group command.

```
SG $D GALE.TECH 7 >SPD>CONSOLE -POOL AB -EFN DEBUGDB
```

Example 2:

Loading Debug by create group request and enter group request commands:

```
CG $D 7 -EFN DEBUGDB

EGR $D GALE.TECH 7 >SPD>CONSOLE -EFN
```

NOTE: The operator terminal is controlled by a system software component called the operator interface manager (OIM) that provides a standard means by which all tasks can communicate with an operator. OIM identifies the messages sent to the operator terminal by providing the task group identification in the prefix to each message; OIM requires you to identify all input by task group. If you are entering Debug directives through the operator terminal, it is recommended that you designate Debug as the OIM default task group; otherwise, each Debug directive must be preceded by Δ$DΔ. To designate Debug as the OIM default task group, enter the following command at any time prior to entering the first Debug directive:

ΔCΔ:$D:

Example 3:

Loading Debug with a directive terminal, not the operator terminal:

```
SG $D GALE.TECH 7 >SPD>KSR01 -EFN DEBUGDB
```

Debug Operation with MMU

The Debug task group is loaded in ring O, a privileged state, in order to run effectively in a protected (MMU) system. The debugger will handle '030F' traps and continue as described below.

An error message will be displayed if the user tries to access non-virtual memory within any debug directive, except the dump memory directive (DP). The debugger will dump as much of the requested memory as possible. Once a non-virtual address is accessed, the rest of the current line to be printed will be blank filled. The current non-virtual address will be advanced to the value that is the next multiple of 1K. This procedure will continue until the area to be dumped is exhausted or the end of memory is reached.

Debug Directives

Debug directives consist of only a directive name or a directive name and one or more arguments. Within a directive, arguments are separated from each other by one or more spaces. Except where specified otherwise, all argument values are entered using hexadecimal notation.

Multiple Debug directives can be entered on a single line. Each directive, except the last, must be followed by a semicolon (;).

Press RETURN at the end of each line (i.e., immediately after the last or only directive).

Symbols used in Debug directive lines are described in Table 5-1.

SECTION 6

MDUMP AND DUMP EDIT UTILITY PROGRAMS

The MDUMP utility program allows a memory dump to be obtained with no requirement that system functions be available. Thus, MDUMP may be used when it is not possible or practical to use the dump facility of debug.

To use MDUMP, you need a disk that contains an MDUMP record on sector 0, and a file (DUMPFILE) to contain the memory dump. Use the create volume command to prepare this disk (see "Preparing for MDUMP," below).

To dump memory to the disk file, bootstrap the prepared disk as described under "Procedure for Using MDUMP," below. This causes the MDUMP record to be loaded and executed. When MDUMP terminates, an image of memory is contained in DUMPFILE. This file can be edited and printed by means of the Dump Edit utility, described later in this section.

MDUMP UTILITY PROGRAM

Preparing for MDUMP

Before loading a program for which a memory dump may be required, enter the create volume command, as follows:

FORMAT:

$$\left\{\begin{matrix}\text{CREATE VOL}\\ \text{CV}\end{matrix}\right\} \text{path} \left\{\begin{matrix}\text{-MDUMP nn}\\ \text{-MD nn}\end{matrix}\right\}$$

ARGUMENT DESCRIPTIONS:

path
: Designates the pathname to the disk volume being prepared for MDUMP. The pathname may be >SPD>sympd or >SPD>sympd>volid. If >volid is specified, the volume label is checked. The volume must have been previously formatted via a create volume command. (This command is described in detail in the *Commands* manual.) The volume can contain other data.

$\left\{\begin{matrix}\text{-MDUMP nn}\\ \text{-MD nn}\end{matrix}\right\}$
: Writes the MDUMP bootstrap record to the volume specified in the path argument and allocates a file (DUMPFILE) large enough to contain nn 4K modules to be dumped. The value of nn should be no larger than the number of 4K modules contained in the system being used.

Procedure for Using MDUMP

Once an executing program encounters a problem or a halt occurs, you can obtain a memory dump by taking the following actions:

1. Bootstrap MDUMP, which then performs the memory dump to the disk file DUMPFILE.
2. Use the Dump Edit utility program to print all or a portion of the memory dump from the disk volume that contains MDUMP'S output.

Procedure For Bootstrapping MDUMP on Non-Model 23 Series Systems

To bootstrap the MDUMP bootstrap record into memory, perform the procedure listed below, MDUMP then transfers to the disk file DUMPFILE the amount of memory image specified in the -MDUMP argument of the create volume command.

1. Mount the disk containing MDUMP on the channel to be used in bootstrapping.
2. Press Stop and CLear.
3. Set the P-register to 0004_{16}.
4. Enter in register B1 the initial address of the memory area into which MDUMP is to be read. MDUMP requires one sector of the disk device type on which it is stored. The initial address of B1 should be greater than 100_{16} to insure that hardware dedicated locations are not overlayed.
5. Enter in register R1 the channel number of the bootstrap device (i.e., the disk mounted in step 1).
6. Press Load, then Execute. The bootstrap record MDUMP is read into the memory location specified in step 4 above, and dumps the amount of memory image that fills DUMPFILE. The dump is complete when an end-of-job halt occurs (see Table 6-1).

NOTE: The size of DUMPFILE is limited by the capacity of the storage device. A maximum of 120K of memory can be stored on a diskette file.

Procedure For Running The QLT And/Or Bootstrapping MDUMP On Model 23 Series 6/20 Systems

1. Press STOP/STEP button.
2. Press MASTER CLEAR button.
3. If QLT, go to Step 5.
4. Set the P register to the first location to transfer the boot prom data. Any value other then zero (prefer 0100 hex).
5. Press LOAD button.
6. Press EXECUTE button.
   CP will stop with P register equal to the initial value plus CA hex.
   B1 register equal to 0100_{16}, and R1 register equal to 0400_{16}.
7. Change R1 if different boot channel desired.
8. Change B1 if different buffer address desired.
9. Press RUN button.
10. Press EXECUTE button.

MDUMP Halts

No messages are issued during execution of MDUMP. If a halt occurs during execution, the contents of the P-register and R6 register must be displayed to determine the significance of the halt, as indicated in Table 6-1.

DUMP EDIT UTILITY PROGRAM

Dumps produced by the Dump Edit utility are written to the user output file which must be capable of receiving a 132-character line.

There are two sources of dumps:

- Files created by the previous execution of the MDUMP utility. (All or selected portions of the file can be dumped.)
- Main memory. (A dump of main memory allows you to determine the configuration under which Dump Edit is executing.)

Dumps produced by dump edit may be logical (edited format) dumps or physical (image format) dumps. Control arguments in the DPEDIT command (described later in this section) allow you to suppress either type of dump. If these control arguments are omitted, execution of Dump Edit produces a full logical dump followed by a full physical dump.

TABLE 6-1. MDUMP HALTS

Register Contents P-register	R6 register	Condition	Operator Action
$003E_{16}$a	=0	End of job	No operator action required. For information only.
$003E_{16}$a	≠0	Disk error	Rebootstrap MDUMP. (R6 contains the disk status word.)
03nn	≠0	Trap handler error has occurred.	For a description of trap messages, see the "Trap Handling" section of the Monitor and I/O Service Calls manual.

[a]Address relative to the initial address of MDUMP as stored in memory.

Logical dumps may be produced for any release of MDT Operating System and for any release of MOD 400 Operating System.

Physical dumps may be produced for any properly prepared dump file, regardless of the content of the dump file.

Logical and physical dumps are printed in both hexadecimal and ASCII notation. Duplicate lines, if any, are suppressed. Suppressed lines are designated as described subsequently under Dump Edit Line Format.

Dump Edit Line Format

The format of a Dump Edit line for both logical and physical dumps is as follows:

Columns	Content
2-6	For LAF: Five hexadecimal digits designating the starting address of the line of dump information; the hexadecimal digit in print position 6 is always 0. This forces the dump line to agree with the template printed at the heading of each page.
3-6	For SAF: Four hexadecimal digits designating the starting address of the line of dump information; the hexadecimal digit in print position 6 is always 0. This forces the dump line to agree with the template printed at the heading of each page.
7	Slash (/)
8-10	Blanks
11-91	Sixteen consecutive words; each word is represented by four hexadecimal digits and is followed by a space.
92-95	Blanks
96-127	ASCII representation of the previous group of 16 consecutive words. A byte that is not printable is designated by a period (.).
1-11	Blanks
12-93	* * * * * * * * * * * *
94-132	Blanks

Physical Dumps

In a physical dump, the leftmost column of data (four hexadecimal digits for SAF; five for LAF) designate real memory addresses. When the Memory management Unit (MMU) is in use, there may be ranges of invalid virtual addresses (discontinuities) in a physical dump from main memory. When an invalid virtual address is encountered, a message within the

physical dump contains the invalid virtual address and the next virtual address to be attempted. Thus, when the physical dump resumes, the valid virtual address is known and the left column continues to designate real memory addresses as if the discontinuity did not exist.

A physical dump from an external dump file does not display invalid virtual address messages, and the left column of addresses is an uninterrupted continuum of physical addresses.

The physical memory dump in Figure 6-1 was produced by Dump Edit in response to the command:

DPEDIT DMPVOL DUMPFILE -NL -TO X'720'

Logical Dumps

A logical dump may be tailored by selecting (or suppressing) task group information on a group identification basis. File system information may also be suppressed. This tailoring is obtained by the use of control arguments in the DPEDIT command.

All addresses in a logical dump are virtual addresses. The leftmost column of data (the addresses whose contents are being shown) are always virtual addresses. This applies to dumps of disk files as well as to dumps of main memory. For disk files, Dump Edit calculates the virtual address in the same way as the Memory Management Unit would under the same conditions.

The arrangement of information in a logical dump is described below and illustrated in Figures 6-2 through 6-4.

SYSTEM SUMMARY

- o Location and contents of hardware-dedicated main storage
- o System Time of Dump
- o Location and contents of System Control Block (SCB)
- o Hardware Configuration
  - Model number of central processor
  - Presence (or absence) of the Commercial Instruction Processor, the Scientific Instruction Processor, and the Memory Management Unit.
  - Value of the real-time clock scan cycle
  - Presence (or absence) of an operator's terminal
  - High address of virtual memory
  - High address of physical memory
- o Software Configuration
  - Name of operating system
  - Presence (or absence) of the error message library
  - Size of trap save area (TSA)
  - Size of interrupt save area (ISA)
  - Number of indirect request blocks (IRBs) in IRB pool
  - Presence (or absence) of the batch task group
- o Batch Group Data (shown if batch group is present)
  - Virtual address of beginning of background
  - Virtual address of the end of background
  - Rollout status (currently rolled out or not)
  - Number of completed rollout/rollin events
  - Size of background memory given to foreground
- o Memory Pool Data[1,2,3]
  - Pool identification
  - Starting address of pool

[1] Supplied for each memory
[2] An "X" appears beside a pool name that can cause the batch group to be rolled out.
[3] The pool name for the batch group is BATCH.

```
MAIN STORAGE DUMP    1978/05/30 0753:16.6      DUMPEDIT-0110-05/25/0748          GCOS6 MOD400-L110-05/18/1052          PAGE 0038

          0    1    2    3    4    5    6    7    8    9    A    B    C    D    E    F

00000/ 0000 FFFF 0000 0000 0400 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000  ................................
00010/ 0000 43A5 0000 0000 0000 E40D 0004 0000 0000 0177 0000 8380 0000 1E53 0000 0000  ..C.............W........S....
00020/ 0000 0000 0000 0001 0000 0101 0000 0103 0000 0105 0000 0107 0000 0109 0000 010B  ................................
00030/ 0000 010D 0000 010F 0000 0111 0000 0113 0000 0115 0000 0117 0000 0119 0000 011B  ................................
00040/ 0000 011D 0000 011F 0000 0121 0000 0123 0000 0125 0000 0127 0000 0129 0000 012B  ...........!...#...%...'...)...+
00050/ 0000 012D 0000 012F 0000 0131 0000 0133 0000 0135 0000 0137 0000 0139 0000 013B  ...-.../...1...3...5...7...9...;
00060/ 0000 013D 0000 013F 0000 0141 0000 0143 0000 0145 0000 0147 0000 0149 0000 014B  ...=...?...A...C...E...G...I...K
00070/ 0000 014D 0000 014F 0000 0151 0000 0153 0000 0155 0000 0157 0000 D151 0000 001B  ...M...O...Q...S...U...W...1....
00080/ 0000 0000 0000 0000 0000 016A 0000 667B 0000 03F9 0000 4C78 0000 064C 0000 0000  ...........J..F.......LX...L....
00090/ 0000 0000 0000 4D2F 0000 4E0F 0000 4EEF 0000 4FCF 0000 50AF 0000 52C1 0000 51AB  ......M/...N...N...O...P...R...Q.
000A0/ 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000  ................................
          *    *    *    *    *    *    *    *    *    *    *    *    *    *    *    *
00100/ 0FFE 8AD3 0F02 8AD3 0F02 8AD3 0F02 8AD3 0F02 8AD3 0F02 8AD3 0F02 8AD3 0F02 8AD3  ................................
00110/ 0F02 8AD3 0F02 8AD3 0F02 8AD3 0F02 8AD3 0F02 8AD3 0F02 8AD3 0F02 8AD3 0F02 8AD3  ................................
          *    *    *    *    *    *    *    *    *    *    *    *    *    *    *    *
00150/ 0F02 8AD3 0F02 8AD3 0F02 8AD3 0F02 8AD3 0F02 8AD3 0F02 8AD3 83C0 16DA 8F78 FFDC  ............................X..
00160/ 4080 9878 FFCF 0001 0000 0001 0C08 8385 0000 0000 0000 0000 0000 0000 0170 4000  @..X.........................P@.
00170/ 0000 0FFF 3035 2F30 332F 3034 3133 0005 FFFC 0000 0000 0000 09DD 0000 6CC3 0000  ....05/03/0413..............L...
00180/ 0000 0000 09B6 0000 6565 0000 09DD 0000 9523 0033 724A 0000 7012 0000 0006 842A  ........EE........#.3RJ..P......*
00190/ 0032 0000 0000 01E0 0000 0000 0000 65B1 0002 7FFF 0068 0025 0000 4E34 0000 09B6  .2............E.......H.%...N4....
001A0/ 0000 0003 0000 0AC9 0011 FE7F 0000 0000 0000 0001 0000 0000 0000 0000 0000 0003  ................................
001B0/ 0000 0000 A033 2442 FFFE 0442 01C1 0049 0000 19DE 0000 0316 0000 1E8C 0003 3AFF  .....3$B...B...I..............:.
001C0/ 0000 5E85 0003 3AFF 0002 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000  ..^...:.........................
001D0/ 0000 0000 0000 0000 0000 0000 0000 0000 0363 2456 FFFF 0000 0000 01EE 0000 0000  ..................C$V...........
001E0/ 0000 0000 049D 0000 13A1 0000 0363 0000 13A7 8047 0000 0006 87E0 0000 0000 0000  ..............C.....G...........
001F0/ 049D 0000 0307 0000 0363 0000 1EFF 0027 0000 01EE 0000 01DA 0000 020A 063B 0002  ..........C.....'..............;..
00200/ 7215 0000 0363 0002 735B 0082 0000 064C 0000 0000 0000 0218 0000 0000 0000 0000  r....C..s[.....L................
00210/ 0000 0000 0000 0000 0000 0000 0000 0000 0000 0226 0000 0000 0000 0000 0000 0000  ....................&...........
00220/ 0000 0000 0000 0000 0000 0000 0000 0234 0000 0000 0000 0000 0000 0000 0000 0000  ................4...............
00230/ 0000 0000 0000 0000 0000 0242 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000  ............B...................
00240/ 0000 0000 0000 0250 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000  .......P........................
00250/ 0000 025E 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 026C 0000  ...^.........................L...
00260/ 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 027A 0000 0000  ..............................Z.....
00270/ 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0288 0000 0000  ................................
00280/ 0000 0000 0000 0000 0000 0000 0000 0000 0000 0296 0000 0000 0000 0000 0000 0000  ................................
00290/ 0000 0000 0000 0000 0000 0000 0000 02A4 0000 0000 0000 0000 0000 0000 0000 0000  ................................
002A0/ 0000 0000 0000 0000 0000 02B2 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000  ................................
002B0/ 0000 0000 0000 02C0 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000  ................................
002C0/ 0000 02CF 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 02DC  ................................
002D0/ 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000  ................................
002E0/ 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 D851 8753 83C0 1ED3 D851 8753  ....................Q.S.....Q.S
002F0/ 83C0 1ED1 B854 83C0 1EBC B854 83C0 09AC B854 83C0 09A7 B854 83C0 0A2D 0000 65A0  .....T.....T.....T.....T...-..e.
00300/ 412A 2A42 5245 414B 2A2A 0000 1EFF 0000 01EE 00C5 FF01 0000 01DA 0000 0177 0001  A**BREAK**.....................w..
00310/ 0000 0000 0000 0000 0000 0000 0001 0000 0000 0000 0001 0000 0000 0000 9D83 0000  ................................
00320/ 0000 0000 0000 0000 0000 0001 0000 0000 0000 0000 0000 0000 0001 0001 0000 0000  ................................
00330/ 033F 0000 0040 0000 0000 0000 0000 0001 0001 0000 0000 0000 0010 0000 0000 4120  .?...@.........................A
00340/ 2020 2020 2020 2020 2020 2020 2020 2020 2020 2020 2020 2020 2020 2020 2020 2020
          *    *    *    *    *    *    *    *    *    *    *    *    *    *    *    *
00360/ 2020 2020 2020 0000 0000 2453 0000 0002 0000 70FB 0000 0000 0000 90C1 0000 632D        ....$S......P...........c-
00370/ 0000 039C 0199 0009 0000 0000 0000 0000 0000 0000 AD31 BDA8 AD70 0000 C7A7 CAC1  .....................1...P.......
00380/ 0000 0000 AD3E 0000 0000 09DD 0000 9F13 0000 9F13 0000 0000 0000 0000 0000 0000  .....>..........................
00390/ 01D2 0000 0000 0000 0000 0000 0000 0000 0000 0000 9FC3 000F 0000 0000 0000 0000  ................................
003A0/ 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000  ................................
003B0/ 0000 0000 0000 0000 0008 0000 0000 0000 0000 0000 0000 9FC3 0000 4C9D 0000 4C78  ..............................L...LX
003C0/ 0000 0563 0000 0000 0001 0000 0000 0000 060A 0000 0000 0000 0000 0000 0000 0000  ...C............................
003D0/ 0000 0000 0000 0002 7F54 0000 0177 0000 49DD 0000 0A79 0000 0000 0000 0000 0000  .........T...w..I....Y..........
```

Figure 6-1. Sample Physical Memory Dump

```
MAIN STORAGE DUMP     1978/05/30 0753:16.6       DUMPEDIT-0110-05/25/0748          GCOS6 MOD400-L110-05/18/1052          PAGE 0039

            0    1    2    3    4    5    6    7    8    9    A    B    C    D    E    F

 003E0/  3648 0000 0000 0000 0000 0000 0000 0363  0000 0000 03F8 0000 03E8 0000 0000 0000   6H..............C...............
 003F0/  03F9 0004 0000 1803 0000 6527 0000 0000  0416 0000 FFFF 0000 0000 1FE7 6004 0000   ...........E'.................@...
 00400/  047C 0000 0177 0000 1FFB 0000 0192 0000  13A1 0000 018D 0000 01E0 001A 842A 0006   .....W.........................*..
 00410/  FFFF FFFF 002D 0000 0000 FF00 0000 0000  0000 0000 0001 0000 0000 0000 0000 0000   .....-..........................
 00420/  0000 0000 0000 0000 0000 0000 0000 0000  0000 0000 0000 0000 0000 0000 0000 0000   ................................
          *    *    *    *    *    *    *    *    *    *    *    *    *    *    *    *
 00440/  0000 0000 0000 544D 0000 09B6 002B 0100  0000 1B1F 0000 1B62 0000 4C9D 0002 7EA8   ......TM.....+.........B..L.....
 00450/  001D 0087 FFFF 0000 0000 0004 0000 0000  0000 0000 A043 0000 2390 0013 0054 0000   ..................C..#.....T..
 00460/  0000 0000 0C59 0000 0000 0000 9ECC 0000  0F18 0000 0177 0000 0000 1CDA 0000 1677   .....Y.................W.........W
 00470/  0000 0000 0000 018D 0000 01E9 0000 201E  0000 0192 0000 1FFB 0000 0000 0000 0000   ..............  ................
 00480/  0000 0000 0000 0000 FFFF FFFF FFFF FFFF  0000 0000 0000 0177 0000 49DD 0000 0000   ........................W..I.....
 00490/  0000 0000 0000 0000 0000 1E3B 0000 0000  0000 0000 0563 0000 0563 0000 0000 049D   ...........;.........C...C......
 004A0/  0000 049D 0000 0000 0000 04AE 0010 0000  0800 0000 0000 0000 0000 04CB 0000 FFFF   ................................
 004B0/  0000 0000 1465 6003 0000 0530 0000 0177  0000 16F8 0000 13A7 0000 0000 0000 0080   .....Ed`....0...W...............
 004C0/  0000 049D 0000 0011 0000 0011 0004 0000  0011 0010 FF00 0000 0000 0000 0000 0001   ................................
 004D0/  0000 0000 0000 0000 0000 0000 0000 0000  0000 0000 0000 0000 0000 0000 0000 0000   ................................
          *    *    *    *    *    *    *    *    *    *    *    *    *    *    *    *
 00510/  0000 0000 0000 0000 0000 0000 0000 0000  0000 0000 0000 0000 0000 0000 0000 0177   ...............................W
 00520/  0000 13A7 0000 0000 0000 0080 0000 65A0  0000 1478 0000 15E6 0000 14EF 0000 16D6   ..............E....X............
 00530/  0000 1E3B 0000 0000 0000 0000 0000 0000  0000 FFFF FFFF FFFF FFFF 0000 0000 0000   ...;............................
 00540/  0177 0000 4ABD 0000 0A79 0000 0000 0000  0000 0000 110F 0000 0000 0000 0000 0000   .W..J....Y......................
 00550/  0000 0363 0000 0000 0552 0000 055C 0000  0000 0000 0563 0010 0000 0800 0000 0000   ...C.....R...R.......C..........
 00560/  0000 0000 0580 0000 FFFF 0000 0000 1465  6003 0000 05E1 0000 0177 0000 16F8 0000   ..............Ed`........W......
 00570/  10E0 0000 0000 0000 0080 0000 055C 0000  2000 2453 0012 0042 0000 0500 0010 FF00   .............R.. .$S...B........
 00580/  0000 0000 0000 0000 0001 0000 0000 0000  0000 0000 0000 0000 0000 0000 0000 0000   ................................
 00590/  0000 0000 0000 0000 0000 0000 0000 0000  0000 0000 0000 0000 0000 0000 0000 0000   ................................
          *    *    *    *    *    *    *    *    *    *    *    *    *    *    *    *
 005B0/  0000 0000 0000 0000 0000 0000 0000 0000  0000 0000 0000 0C59 0000 1925 000B 0000   .......................Y...%....
 005C0/  09B6 00BA 00BF 0000 1925 001C 0000 0000  0C59 0000 0000 0000 9868 0000 0F18 0000   .........%.......Y.......H......
 005D0/  0177 0000 10E0 0000 10CF 0000 10CF 0000  4ABD 0000 1478 0000 10CF 0000 14EA 0000   .W..............J....X..........
 005E0/  16D6 0000 1186 0000 105F 0000 4C78 0000  0000 0000 0000 0000 0000 0000 0000 0000   ..........._..Lx................
 005F0/  0000 0000 0000 1659 0000 2E74 0000 0FB9  0000 4A64 0000 15E6 0000 14EF 0000 16D6   .......Y...T......JD............
 00600/  0000 0F6F 0000 06DA 0005 0000 FFFE 0000  0000 064C 1400 2333 0400 8000 0000 0000   ...O.............L..#3..........
 00610/  0003 18FD 99CD 0000 2E0F 0000 1358 0030  0018 FFFF 0000 0000 0000 0000 0000 0000   ..............X.O...............
 00620/  0000 0000 FFFF FFFF FFFF FFFF 0000 0000  0000 0177 0000 4A64 0000 0000 0000 0000   ...................W..JD..........
 00630/  0000 0000 0000 0F6F 0000 0000 0000 0000  0000 0000 0363 0000 0000 063B 0000 063B   .......O.............C.....;...;
 00640/  0000 0000 0000 064C 0006 0000 0801 0000  0000 0000 0000 0000 1406 FFFF 0000 0000   .......L........................
 00650/  1465 6003 0000 0600 0000 0177 0000 16F8  0000 0A9D 0000 2E74 0000 0020 0000 063B   .Ed.........W...........T... ...;
 00660/  0018 0000 8000 0000 1400 0000 0000 0006  FF00 0000 0000 0000 0000 0001 0000 0000   ................................
 00670/  0000 0000 0000 0000 0000 0000 0000 0000  0000 0000 0000 0000 0000 0000 0000 0000   ................................
          *    *    *    *    *    *    *    *    *    *    *    *    *    *    *    *
 006D0/  0000 0000 0000 0000 8000 0000 0000 06DA  4000 FF00 0000 0FFF 0000 0000 3F02 0001   ................@...........?...
 006E0/  0001 80C1 0000 6AA6 0000 6AA8 0000 66A9  0000 90C3 0000 6629 0000 696D 0000 6A19   ......J...J...F.......F)..IM..J.
 006F0/  0000 66A3 0000 4143 FF00 0000 0000 0000  0000 0000 0000 0000 0000 1B62 0000 0FED   ..F...AC.....................B..0.
 00700/  0000 0000 002D FFFF 0011 0041 0700 0000  0000 0C59 0000 1925 000B 0000 09B6 00BA   ..... .....A........Y...%........
 00710/  00BF 0000 1B1F 0000 1B62 0000 0000 5594  0000 09B6 00E0 00EF 0000 1B1F 0000 1B62   .........B....U.................B
 00720/  0000 7024 0000 4A07 0023 0011 00E0 0002  0005 0042 030F 0000 7024 0000 6FE3 0000   ..P$..J..#.........B....P$..U...
RDY:
```

Figure 6-1 (cont). Sample Physical Memory Dump

```
MAIN STORAGE DUMP     1978/05/30 0753:16.6       DUMPEDIT-0110-05/25/0748          GCOS6 MOD400-L110-05/18/1052          PAGE 0001

           0    1    2    3    4    5    6    7     8    9    A    B    C    D    E    F

HARDWARE  DEDICATED LOCATIONS
 00000/   0000 FFFF 0000 0000 0400 0000 0000 0000   0000 0000 0000 0000 0000 0000 0000 0000   ................................
 00010/   0000 43A5 0000 0000 0000 E40D 0004 0000   0000 0177 0000 8380 0000 1E53 0000 0000   ..C.................W........S....
 00020/   0000 0000 0000 0001 0000 0101 0000 0103   0000 0105 0000 0107 0000 0109 0000 010B   ................................
 00030/   0000 010D 0000 010F 0000 0111 0000 0113   0000 0115 0000 0117 0000 0119 0000 011B   ................................
 00040/   0000 011D 0000 011F 0000 0121 0000 0123   0000 0125 0000 0127 0000 0129 0000 012B   ...........!...#...%...'...)...+
 00050/   0000 012D 0000 012F 0000 0131 0000 0133   0000 0135 0000 0137 0000 0139 0000 013B   ...-.../...1...3...5...7...9...;
 00060/   0000 013D 0000 013F 0000 0141 0000 0143   0000 0145 0000 0147 0000 0149 0000 014B   ...=...?...A...C...E...G...I...K
 00070/   0000 014D 0000 014F 0000 0151 0000 0153   0000 0155 0000 0157 0000 0151 0000 001B   ...M...O...Q...S...U...W...1....
 00080/   0000 0000 0000 0000 0000 016A 0000 667B   0000 03F9 0000 4C78 0000 064C 0000 0000   ...........J..F........LX...L....
 00090/   0000 0000 0000 4D2F 0000 4E0F 0000 4EEF   0000 4FCF 0000 50AF 0000 52C1 0000 51AB   ......M/..N...N...O...P...R...Q.
 000A0/   0000 0000 0000 0000 0000 0000 0000 0000   0000 0000 0000 0000 0000 0000 0000 0000   ................................
           *    *    *    *    *    *    *    *     *    *    *    *    *    *    *    *
 00100/   0FFE 8AD3 0F02 8AD3 0F02 8AD3 0F02 8AD3   0F02 8AD3 0F02 8AD3 0F02 8AD3 0F02 8AD3   ................................

SYSTEM TIME OF DUMP       1901/01/01 0007:07.0

SYSTEM CONTROL BLOCK    00177
 00170/   0000 0FFF 3035 2F30 332F 3034 3133 0005   FFFC 0000 0000 0000 09DD 0000 6CC3 0000   ....05/03/0413..............L...
 00180/   0000 0000 09B6 0000 6565 0000 09DD 0000   9523 0033 724A 0000 7012 0000 0006 842A   ........EE.......#.3RJ..P.......*
 00190/   0032 0000 0000 01E0 0000 0000 0000 65B1   0002 7FFF 0068 0025 0000 4E34 0000 09B6   .2.............E......H.%..N4....
 001A0/   0000 0003 0000 0AC9 0011 FE7F 0000 0000   0000 0001 0000 0000 0000 0000 0000 0003   ................................
 001B0/   0000 0000 A033 2442 FFFE 0442 01C1 0049   0000 19DE 0000 0316 0000 1E8C 0003 3AFF   .....3$B...B...I..............:.
 001C0/   0000 5E85 0003 3AFF 0002 0000 0000 0000   0000 0000 0000 0000 0000 0000 0000 0000   ..^...:.........................
 001D0/   0000 0000 0000 0000 0000 0000 0000 0000   0363 2456 FFFF 0000 0000 01EE 0000 0000   ..................C$V.............

CENTRAL PROCESSOR MODEL: 4X OR 5X
COMMERCIAL PROCESSOR:  NO
SCIENTIFIC PROCESSOR:  NO
MEMORY MANAGEMENT UNIT IN USE:   YES
TIME (MILLI-SEC) BETWEEN REAL TIME CLOCK INTERRUPTS: 0032
OPERATOR'S TERMINAL: YES
HIGH PHYSICAL MEMORY ADDRESS: 27FFF
HIGH VIRTUAL  MEMORY ADDRESS: 33AFF

OPERATING SYSTEM:  MOD400/L110
ERROR MESSAGE LIBRARY: YES
SIZE OF TRAP SAVE AREA (WORDS):  0068
SIZE OF INTERRUPT SAVE AREA (WORDS): 0025
NUMBER OF UNUSED INDIRECT REQUEST BLOCKS:  0033

BATCH GROUP: YES
VIRTUAL ADDRESS OF BEGINNING OF BACKGROUND: 30000
VIRTUAL ADDRESS OF THE END   OF BACKGROUND: 33AFF
CURRENTLY ROLLED-OUT:  NO
NUMBER OF COMPLETED ROLL-OUT/ROLL-IN EVENTS: 0001
MEMORY GIVEN TO FOREGROUND FROM BACKGROUND (WORDS):  0000
```

Figure 6-2. Logical Dump: System Summary

```
MAIN STORAGE DUMP     1978/05/30 0753:16.6       DUMPEDIT-0110-05/25/0748          GCOS6 MOD400-L110-05/18/1052          PAGE 0002

           0    1    2    3    4    5    6    7    8    9    A    B    C    D    E    F

MEMORY POOL DEFINITIONS                          MEMORY POOLS STATUS
  POOL       START      END       SIZE     TOTAL UNUSED     MAXIMUM UNUSED     NUMBER OF   NUMBER
  NAME      ADDRESS    ADDRESS  (WORDS)      (WORDS)     CONTIGUOUS (WORDS)  FRAGMENTS OF USERS
   $$        065C0      0A05F    03AA0        02420            01F40            00007      00001
   BATCH     30000      33A3F    03A40        01C40            01C40            00001      00001
   TS   X    0A060      0AE1F    00DC0        00DC0            00DC0            00001      00000
   EX   X    0AE20      0C79F    01980        01980            01980            00001      00000
   AB        0C7A0      244DF    17D40        16B60            16AE0            00002      00002


SYSTEM SYMBOL TABLE
BOUND UNIT    SYMBOL      VALUE
 Z3EXEC
 FILBUF       FILBUF      00000
 ROLLCD
```

Figure 6-2 (cont). Logical Dump: System Summary

```
MAIN STORAGE DUMP    1978/05/30 0753:16.6      DUMPEDIT-0110-05/25/0748          GCOS6 MOD400-L110-05/18/1052          PAGE 0003

          0   1   2   3   4   5   6   7     8   9   A   B   C   D   E   F

TREE OF VOLUME, DIRECTORY AND FILE DESCRIPTOR BLOCKS


VOLUME DESCRIPTOR BLOCK
DEPTH  00      LOCATION 009DD
*****************************
 009D0/   4E45 5957 454C 4C49 4E46 4F52 4D41 5449   4F4E 5359 5354 454D 5300 0000 4054 0000    NEYWELLINFORMATIONSYSTEMS...MT..
 009E0/   0000 0000 0A26 0000 0000 0000 0000 0001   0000 0000 0000 0026 0000 0000 0000 0001    .....&.................&........
 009F0/   0001 0000 0000 0000 5A53 5953 3531 2020   2020 2020 2620 0020 0100 FFFF 000B 000D    ........ZSYS51      & . ........
 00A00/   0000 0000 0000 0000 0000 0000 0000 0000   0000 0000 0000 0000 0000 0000 0000 0000    ................................
 00A10/   0000 0000 0000 0000 0100 0108 0001 0000   000D 0100 0000 000D 0000 0000 0000 0000    ................................
 00A20/   0009 0200 0000 5243 4430 3020 0000 0AA7   0000 09DD 0000 0000 0000 0000 0000 0000    ......RCD00 ....................

          FILE DESCRIPTOR BLOCK
          DEPTH  01      LOCATION 00A26
          *****************************
 00A20/   0009 0200 0000 5243 4430 3020 0000 0AA7   0000 09DD 0000 0000 0000 0000 0000 0000    ......RCD00 ....................
 00A30/   0001 0000 0000 0000 0100 0000 0000 0000   0001 0002 0000 0000 0000 5A33 4558 4543    ..........................Z3EXEC
 00A40/   5554 4956 454C 0782 0100 0100 FFFF 0018   01E0 0000 0000 0000 0000 0000 0000 0000    UTIVEL..........................
 00A50/   0000 0000 0000 0000 0000 0000 0000 0000   0000 0000 0000 0000 0000 0100 0108 0001    ................................
 00A60/   0000 01CC 0100 0000 01E0 0000 0000 0000   0000 2020 C000 0000 56E1 01EA 0100 0000    .................. ....V........

          DIRECTORY DESCRIPTOR BLOCK
          DEPTH  01      LOCATION 00AA7
          *****************************
 00AA0/   0102 0000 56E1 01EA 0111 0000 0000 0000   9F43 0000 09DD 0000 4C9D 0000 0000 0000    ....V...........C.......L.......
 00AB0/   0000 0001 0000 0000 0000 000A 0000 0000   0000 0001 0001 0000 0000 0000 5350 4420    ............................SPD
 00AC0/   2020 2020 2020 2020 2020 0020 0100 0000   0000 0009 0000 0000 0000 0140 0000 5596              . ..............@..U.

                    FILE DESCRIPTOR BLOCK
                    DEPTH  02      LOCATION 04C9D
                    *****************************
 04C90/   0500 0000 0000 0005 FF00 FF00 FF00 FF03   FF20 FF00 FF00 0000 0000 0000 51D0 0000    ..................... .........Q...
 04CA0/   0AA7 0000 0000 0010 0000 0000 0000 0001   0000 0000 0000 0008 0000 0000 0000 0000    ................................
 04CB0/   0030 0000 0000 0000 434F 4E53 4F4C 4520   2020 2020 4312 0087 0087 0000 4D2F 1489    .0......CONSOLE      C........M/..

                    FILE DESCRIPTOR BLOCK
                    DEPTH  02      LOCATION 051D0
                    *****************************
 051D0/   0000 52E6 0000 0AA7 0000 0000 000E 0000   0000 0000 0001 0000 0000 0000 2442 0101    ..R.....................$B..
 051E0/   0000 9903 0005 0000 0000 0000 0000 4C50   5430 3020 2020 2020 2020 0110 0089 0089    ..............LPT00         ......
 051F0/   5C00 82D7 0003 8A55 8000 4E11 D3C0 0055   190A 5E08 5055 9870 00D0 9544 0007 1005    \......U..N....U..^.PU.P...D....

                    BUFFER CONTROL BLOCK    09903
 09900/   0001 0000 9B41 0000 0000 0000 0000 4255   0000 005E 0000 9583 0084 0089 0000 0001    .....A.........BU...>............
 09910/   0000 0001 0000 0000 0000 0000 4A15 0041   0501 0000 9583 0084 0000 0000 0000 2020    ............J...A................

                    FILE DESCRIPTOR BLOCK
                    DEPTH  02      LOCATION 052E6
                    *****************************
```

Figure 6-3. Logical Dump: Tree of File System Structures

```
MAIN STORAGE DUMP     19/8/05/30 0/53:16.6       DUMPEDIT-0110-05/25/0748          GCOS6 MOD400-L110-05/18/1052          PAGE 0004

            0    1    2    3    4    5    6    7     8    9    A    B    C    D    E    F

052E0/   FF03 FF20 FF00 FF00 0000 0000 0000 0000   0000 0AA7 0000 0000 0000 0000 0000 0000   ... ..............................
052F0/   0001 0000 0000 0000 0000 0000 0000 0000   00FC 0000 0000 0000 0000 4344 5230 3020   ..........................CDR00
05300/   2020 2020 2020 0211 0050 0050 0021 0000   0000 544D 0000 1830 E952 0281 00AE B857           ...P.P.!....TM...0.R.....W

         DIRECTORY DESCRIPTOR BLOCK
         DEPTH  01       LOCATION 09F43
         ****************************
09F40/   0003 0000 9EE1 0000 9C43 0000 09DD 0000   9CC3 0100 0000 0000 0000 0001 0000 0000   .........C........................
09F50/   0000 0002 0000 0000 0000 0001 000E 0000   0000 0000 5349 4420 2020 2020 2020 2020   ......................SID
09F60/   2620 0020 0100 FFFF 0280 0008 0000 0000   0000 0000 0000 0000 0000 0000 0000 0000   & . ..............................
09F70/   0000 0000 0000 0000 0000 0000 0000 0000   0000 0000 0100 0108 0001 0000 0008 00C0   ................................
09F80/   0000 0008 0000 0000 0000 0000 0000 0000   0000 0000 0000 0000 0000 0000 0000 0000   ................................

                  FILE DESCRIPTOR BLOCK
                  DEPTH  02       LOCATION 09CC3
                  ******************************
09CC0/   0003 0000 9F41 0000 0000 0000 9F43 0000   0000 0900 0000 0000 0000 0001 0000 0000   .....A........C..................
09CD0/   0000 0100 0001 0000 0000 0001 003E 0000   0000 0000 524F 4C4C 4F55 5420 2020 2020   ..............>.......ROLLOUT
09CE0/   0702 0100 0100 FFFF 2418 0078 0000 0000   0000 0000 0000 0000 0000 0000 0000 0000   ........$..X.....................
09CF0/   0000 0000 0000 0000 0000 0000 0000 0000   0000 0000 0100 0108 0001 0000 0076 0100   ............................V..
09D00/   0000 0078 0000 0000 0000 0000 0000 0000   0000 0000 0000 0000 0000 0000 0000 0000   ...X............................

         DIRECTORY DESCRIPTOR BLOCK
         DEPTH  01       LOCATION 09C43
         ****************************
09C40/   0003 0000 A001 0000 9523 0000 09DD 0000   9BE3 0100 0000 0000 0000 0001 0000 0000   .........#........................
09C50/   0000 0001 0000 0000 0000 0001 0012 0000   0000 0000 454D 4C20 2020 2020 2020 2020   ....................EML
09C60/   2620 0020 0100 FFFF 1370 0008 0000 0000   0000 0000 0000 0000 0000 0000 0000 0000   & . .....P.......................
09C70/   0000 0000 0000 0000 0000 0000 0000 0000   0000 0000 0100 0108 0001 0000 0001 0080   ................................
09C80/   0000 0008 0000 0000 0000 0000 0000 0000   0000 0000 0000 0000 0000 0000 0000 0000   ................................

                  FILE DESCRIPTOR BLOCK
                  DEPTH  02       LOCATION 09BE3
                  ******************************
09BE0/   0003 0000 9C41 0000 9B83 0000 9C43 0000   0000 0900 0000 0000 0000 0001 0000 0000   .....A........C..................
09BF0/   0000 0100 0001 0000 9D63 0001 0001 0002   0000 0000 454D 4C46 494C 4520 2020 2020   .........C..........EMLFILE
09C00/   4FB3 0080 0100 FFFF 1378 00A0 0000 0000   0001 0000 FFFD 0014 0003 0000 0000 0000   O........X......................
09C10/   0002 0000 0000 009A 0000 0001 0000 0000   0000 0000 0100 0108 0001 0000 0099 0100   ................................
09C20/   0000 00A0 0000 0000 0000 9D85 0000 0000   0000 0000 0000 0000 0000 0000 0000 0000   ................................

                  BUFFER CONTROL BLOCK    09D63
09D60/   0001 0000 9A41 0000 0000 0000 0000 0000   0000 0005 0000 9A43 0100 0000 0004 0000   .....A..................C........
09D70/   0000 0001 0000 0000 0000 0000 4A23 0001   0102 0000 9A43 0100 137C 0000 0000 0000   ..............J#........C........

                  FILE DESCRIPTOR BLOCK
                  DEPTH  02       LOCATION 09B83
                  ******************************
09B80/   0003 0000 9BE1 0000 0000 0000 9C43 0000   0000 0900 0000 0000 0000 0001 0000 0000   ..............C.................
09B90/   0000 0000 0000 0000 9E03 0001 0003 0004   0000 0000 454D 4C46 494C 4520 2020 2020   ....................EMLFILE
09BA0/   4FB4 0080 0100 FFFF 1418 0008 0000 0000   0003 0000 FFFD 0001 0001 0000 0000 0000   O................................
```

Figure 6-3 (cont). Logical Dump: Tree of File System Structures

```
MAIN STORAGE DUMP    1978/05/30 0753:16.6      DUMPEDIT-0110-05/25/0748          GCOS6 MOD400-L110-05/18/1052          PAGE 0007

          0   1   2   3   4   5   6   7     8   9   A   B   C   D   E   F

TASK GROUP STRUCTURES FOR GROUP $B
**********************************

GROUP CONTROL BLOCK    06CC3
06CC0/   0013 0082 0001 0000 CC23 2442 0030 44U1   0000 6FBB 0000 6F5B 0003 0101 0000 6CF7   .........#$B.0D...O...Uf......L.
06CD0/   0000 6EF4 0001 0001 0000 0000 0000 0000   0000 0001 5780 0000 0000 0000 0CB0 0000   ..N..................W...........
06CE0/   0000 0000 0000 0000 0000 09DD 0003 39D3   0003 3993 0003 0003 0000 0000 0000 0000   ..............9...9.............
06CF0/   0000 0000 0000 0000 0000 0000 00FC 0000   0000 0000 0000 0000 0000 0000 0000 0000   ................................
06D00/   0000 0000 0000 0000 0000 0000 0000 0000   0000 0000 0000 0000 0000 0000 0000 0000   ................................
           *    *    *    *    *    *    *    *     *    *    *    *    *    *    *    *
06EF0/   0000 0000 0000 000F 0000 0000 0000 0000   0000 0000 0000 0000 0000 0000 0000 0000   ................................
06F00/   0000 0000 0000 0000 0000 0000 0000 0000   0000 0000 0000 0000 0000 0000 0000 0000   ................................
06F10/   0000 0000 0000 0000 0000 0000 0000 0000   0000 0000 0000 0000 0000 0000 0000 0000   ................................

         TASK CONTROL BLOCK    06FBB      DPEDIT
06F80/   0003 0000 9FA1 0000 6FBB 0031 0000 0000   0000 0000 0000 0003 3A13 0003 3A13 0003   ........U..1..........:...:...
06F90/   3A13 4001 0000 0000 4000 0003 016L 0000   0177 0000 9903 0000 0A79 0000 0000 0003   :.@.....@....L...W.......Y......
06FA0/   012A 0003 0151 0000 FFFF 0000 0000 6F5B   0000 6CC3 0000 0000 4A77 0000 4A77 0000   .*...Q.........Uf..L.....JW..JW..
06FB0/   0000 0000 6FBB 0031 0000 7402 0000 0000   00B7 0000 45AD 0000 FFFF 0003 0000 2C3C   ....U..1..T.........E..........,<
06FC0/   6003 0000 93F6 0000 0177 0000 1794 0000   9912 0000 9903 0000 51D0 0000 9903 0020   @.........W.............Q.......
06FD0/   003E 0000 0084 0001 FFFF FFFF 0000 FF00   FF00 FF00 FF03 FF20 FF00 FF00 0000 0000   .>.................. ........
06FE0/   0001 0000 9581 030F 4A07 0F01 FE00 6F5B   0000 0001 0000 6FEC 0004 4143 5442 2020   ........J.....Uf......U...ACTB

                IND  REQUEST BLOCK    04A77
04A70/   0000 0000 0002 0000 CA3B 0000 0000 FFFF   0000 6FAA 0003 00C3 0000 6CC3 0000 4A2B   .........:........U.......L...J+
04A80/   000B 0000 6FBB 0000 0000 FFFF 0000 6F4A   0000 7063 0000 6CC3 0000 0000 0002 0000   ....U.........UJ..PC..L.........

                TASK REQUEST BLOCK    300C3
300C0/   0002 0003 3981 0000 4A77 00C1 FE00 0000   0000 0003 0003 00D0 0003 00D5 0003 00D9   ....9...JW......................
300D0/   0006 4450 4544 4954 2020 0004 2D4D 4540   2020 0003 2D4E 4C20 0102 0003 00C3 0003   ..DPEDIT  ..-MEM  ..-NL ........
300E0/   0003 0003 3981 5244 4E5B 0001 FE00 0000   0000 0002 0003 00EE 0003 00F1 0003 4357   ....9.RDN[.....................CW

                TRAP SAVE AREA     045AD
         INSTRUCTION WHICH TRAPPED IS AN MLL AT LOCATION 3158E.  FUNCTION CODE IS 0B01.
         INSTRUCTION: 0001    P-COUNTER: 31590  I': 5F24   Z: 8001     A:  3158E  R3: 0009   B3: 312A5
045A0/   0009 000F 0000 1887 1264 0000 0325 0000   0177 0000 186F 030F 0000 0000 0000 5F24   .........D...%...W...O........?$
045B0/   0009 0001 8001 0003 158E 0003 1590 0003   12A5 0000 9303 0000 03E0 0000 03E0 0003   ................................
045C0/   164F 0003 12F9 0001 0061 FF00 0000 0000   0000 0000 0000 0000 0000 0000 0000 0000   .O.......A......................
045D0/   0000 0000 0000 0000 0000 0000 0000 0000   0000 0000 0000 0000 0C59 0000 0000 254D   .........................Y......%M
045E0/   0000 285F 0003 12A5 0000 9903 0045 0000   0084 5980 0046 0000 0025 0000 9903 0000   ..(?.........E....Y..F...%......
045F0/   51D0 0003 3983 0003 3993 0000 2246 0000   2BA7 0000 0000 2111 0000 21BD 0003 12A5   Q...9...9..."F..+.....!...!.....
04600/   0003 12A5 0000 21B0 0000 0084 0001 0001   0000 FFFF 0000 1D80 0000 21B0 0003 3993   ......!.........A..........!...9.
04610/   0000 0084 0000 1E74 0000 0000 0000 5F22   0001 0001 8001 0000 6AA6 0000 6AA8 0003   .......T.......?".........J...J...

         TASK CONTROL BLOCK    06F5B      EC
06F20/   0003 0000 6F04 0000 0000 0000 0000 0000   0000 0000 0000 0003 3A13 0003 3A13 0003   ....U...................:...:...
06F30/   3A13 0000 0000 0000 0000 0000 0000 0003   0003 0000 49F9 0003 00E1 0000 0000 0000   :.....................I...........
06F40/   695D 0000 69A7 0000 0000 0000 0000 0000   0000 6CC3 0000 0000 4A85 0000 4A85 0000   i]..i.............L.....J...J...
06F50/   0000 0000 6F5B 0030 0000 54B6 0000 0000   0049 0000 4615 0000 FFFF 0003 0000 1465   ....Uf.0..T......I..F...........E
06F60/   6003 0000 9A19 0000 0177 0000 17D0 0000   4A2D 0000 0000 0000 0080 0000 49EB 0000   @.........W.......J-..........I...
```

Figure 6-4. Logical Dump: Task Group Structures

```
MAIN STORAGE DUMP     1978/05/30 0753:16.6      DUMPEDIT-0110-05/25/0748          GCOS6 MOD400-L110-05/18/1052          PAGE 0008

          0    1    2    3    4    5    6    7     8    9    A    B    C    D    E    F

06F70/  1879 0031 0089 0001 0000 FFFF 0030 FF00  FF00 FF00 FF03 FF20 FF00 FF03 0000 0000   .Y.1.........0.........  ........
06F80/  0003 0000 9FA1 0000 6FBB 0031 0000 0000  0000 0000 0000 0003 3A13 0003 3A13 0003   ........O..1..........:...:...


                IWD  REQUEST BLOCK     04A85
04A80/  000B 0000 6FBB 0000 0000 FFFF 0000 6F4A  0000 7063 0000 6CC3 0000 0000 0002 0000   ....O.........OJ..PC..L.........
04A90/  6F5B 0000 0000 0000 0000 0000 0000 0000  0000 CC23 0000 D7BB 0021 0000 4A93 0000   O[..................#.....!..J...


              BATCH REQUEST BLOCK     07063
07060/  0003 0000 0000 0000 4A85 00C1 0000 0000  0000 0004 0000 7072 0000 7075 0000 0000   ........J.............PR..PU....
07070/  0000 7062 0004 422F 3120 0014 5E5A 5359  5335 313E 5350 443E 434F 4E53 4F4C 4520   ..P...B.1 ..^ZSYS51>SPD>CONSOLE
07080/  2020 2020 000C 5E5A 5359 5335 313E 4849  5320 0002 0000 708E 0000 708F 0000 000C       ..^ZSYS51>HIS ....P...P.....
07090/  3E53 5044 3E43 4F4E 534F 4C45 2020 2020  2020 2020 2020 2020 2020 2020 2020 2020   >SPD>CONSOLE
070A0/  0000 0000 0000 0000 0000 0000 0000 0000  0000 0000 0000 0000 0000 0000 0000 0000   ................................
070B0/  0000 0000 0000 0000 0000 0000 0000 0000  0000 0000 0000 0000 0000 0000 0000 0000   ................................


              TRAP SAVE AREA     04615
        INSTRUCTION WHICH TRAPPED IS AN MCL AT LOCATION 06AA6.  FUNCTION CODE IS 0C08.
        INSTRUCTION: 0001    P-COUNTER: 06AA8  I': 3F22   Z: 8001      A:  06AA6  R3: 0001   B3: 30029
04610/  0000 0064 0000 1E74 0000 0000 0000 3F22  0001 0001 8001 0000 6AA6 0000 6AA8 0003   .......T......?"........J...J...
04620/  0029 0000 9923 0000 7089 0000 0000 0000  6A19 0003 0023 0000 4450 FF00 0000 0000   .)...#..P.......J....#..DP......
04630/  0000 0000 0000 0000 0000 0000 0000 0000  0000 0000 0000 0000 0000 0000 0000 0000   ................................
04640/  0000 0000 0C59 0000 1925 000B 0000 09B6  00BA 00BF 0000 1B1F 0000 1B62 0000 0000   .....Y...%.................B....
04650/  5594 0000 09B6 00E0 00EF 0000 1B1F 0000  1B62 0003 0151 0000 4A77 1879 0031 00E0   U.................B...Q..JW.Y.1..
04660/  0002 0005 004C 0000 0003 0151 0003 00C3  0000 5D48 0003 0029 0029 0000 0003 00C3   .....L.....Q.......]H...).).....
04670/  000F FFFF 0003 00DC 000F 0008 000F 0003  0003 0000 0000 1E74 0000 0000 0000 3F22   ........................T.......?"


        WORK SPACE BLOCK      30101
30100/  00E0 0003 3941 4450 4544 4954 0000 0004  0000 1B80 0000 0000 2001 0001 0017 1879   ....9ADPEDIT............. ......Y
30110/  0000 0000 0001 0033 E787 0337 0000 1879  2001 003A E787 0111 0000 1879 2001 003E   .......3...7...Y ..:........Y ..>
30120/  E787 02BF 0000 1879 0001 0045 E787 01A5  0000 1879 0000 0000 0001 2001 0003 3953   .......Y...E.......Y...... ...9S
30130/  0003 012A 0003 0103 0003 013A 0000 0000  0000 0000 0017 2041 4450 4544 4954 2D30   ...*.......:.........  ADPEDIT-0
30140/  3131 302D 3035 2F30 352F 3039 3132 000E  4144 5540 5020 434F 4D50 4C45 5445 0015   110-05/05/0912..ADUMP COMPLETE..
30150/  0C00 BE53 DFC0 0100 FFC0 00FC CBC0 0015  0001 0A00 1981 151D 2C01 0001 0A01 1981   ...S.....................,.....
30160/  1518 2C31 0001 0A01 1981 1513 2C0F 0001  0A01 1981 150E 0F89 3D0F 0901 0F36 8DF3   ..,1.....,.....=....6..
30170/  CBC0 1510 CF83 0003 CBC0 FFC6 7C01 E844  FFFF 0001 0803 CBC0 1465 0001 1404 A870   .............D.....E.....P
30180/  2020 AF40 145F 9BC0 FFBE ABC0 1452 1CFB  F871 FF72 17FE 93C0 007A 93C0 0D50 82C0     .@._.....R...Q.R.....Z...]..
30190/  FFBF 0004 0587 9C80 0000 0018 CCC1 0021  0F88 9CC0 0253 8DD1 0901 00A2 93C0 01F1   .............!.....S.....
301A0/  CFC0 00AE CFC0 0EAF CB80 0000 0000 CFC0  0EAC 0001 0506 CBC0 1418 5C14 0001 0504   ..........................\.....
301B0/  93C0 14EB 95C0 0D49 82C0 FF9A 0001 0515  93C0 02B0 93C0 02C2 93C0 02EC 93C0 0D54   .......I.....................T
301C0/  93C0 0D67 95C0 0D5C 82C0 FF8A 0040 0503  93C0 0D68 93C0 036A 82C0 FF82 0002 0503   ...G...\.....@.....H...J........
301D0/  93C0 0DCA 85C0 14AE 0000 0000 0000 0000  0000 0000 0000 0000 0000 0000 0000 0000   ................................
301E0/  0000 0000 0000 0000 0000 0000 0000 0000  0000 0000 0000 0000 0000 0000 0000 0000   ................................
          *    *    *    *    *    *    *    *     *    *    *    *    *    *    *    *
30200/  0000 0000 0000 0000 0000 0000 9FC0 0041  BCC0 0048 9873 1D01 0381 0032 1EFF ECC3   ...............A...H.S.....2....
30210/  0002 B876 2C00 F0A6 7D2D 090D ABC0 01E2  AFC0 01D4 F0A6 F7EE 3EFF 39FD 1EFF 1909   ...V,....-.............>.9.....
30220/  8AC0 0029 9F40 0032 0F86 9840 002F 1983  83C8 001F A840 001F ACEF ECD3 AF40 001B   ...).@.2...@./.......@........@..
30230/  93C0 0028 3901 0008 B8C0 FF05 93B3 BCD6  88C0 001C 0FEB 9870 2507 0F81 143A 9870   ...(9.............P%.....:.P
30240/  2512 0FFC 9870 2503 0FF9 9870 2502 0FF6  0003 018D 0003 0000 0000 0002 7FFF 0002   %:...P%....P%..................
30250/  7FFF 0003 00C9 0000 16CA 0000 0003 00CA  0003 9FC0 0028 F872 BFC0 FFF9 AF40 FFF9   .....................(.R.....@..
30260/  9BC0 0027 0F83 9CC0 0020 B871 9853 1041  8ED1 1E01 CB91 CFC0 0018 390D B957 0902   ...'..... .Q.S.A..........9..W..
30270/  0FF3 1C00 2C00 C0DD D0EE C955 09ED 88D3  3904 0FFA 8755 0F85 ABC8 0007 B842 FFFF   ....,.......U....9....S.......B..
30280/  83C8 0001 0003 0232 0003 029C 0000 0000  000B 2D4E 4F5F 4C4F 4749 4341 4C20 019A   .......2..........-NO_LOGICAL ..
```

Figure 6-4 (cont). Logical Dump: Task Group Structures

- End address of pool
- Total size of pool
- Total available space
- Maximum contiguous available space
- Number of available fragments (pieces) of pool space
- Number of users

o System Symbol Table
The names and values of all symbols that have an entry in the system symbol table are displayed. Symbols are grouped according to the bound unit(s) in which they occur.

File System Structures

The logical dump displays the location and content of the following file system structures:

- o Volume Descriptor Blocks (VDBs)
- o Directory Descriptor Blocks (DDBs)
- o File Descriptor Blocks (FDBs)
- o Buffer Control Blocks (BCBs)

The hierarchy of these structures is indicated by the dump as shown in Figure 6-2, which is an abridged section of a logical dump. Each block is assigned an integer that corresponds to the level of the block in the hierarchy. The headings of all blocks are indented according to the depth of the block. This makes it easy to see which files belong to volume major directories and which belong to subordinate directories.

The display of the tree of file system structures may be suppressed at run-time.

TASK GROUP STRUCTURES

The logical dump displays the location and content of the following structures as shown by Figure 6-3, which is an abridged section of a logical dump.

- o Group Control Blocks (GCBs)
- o Task Control Blocks (TCBs) for each GCB
- o Indirect Request Blocks (IRBs) for each TCB
- o Request Blocks (Group, Task, or I/O) for each IRB
- o Trap Save Areas (TSAs) for each TCB
- o File Control Blocks (FCBs) for each GCB
- o Work Space Blocks for each GCB

At run-time, the task group structures for any designated task group may be displayed or the display may be suppressed.

For the system task group, IRBs (and hence also RBs) are displayed only when the file being processed is an external dump file; i.e., the display is suppressed when the input is from current main memory.

Work space blocks and FCBs for the batch task group are not displayed when the batch group is rolled out.

The display of structures for each task group is preceded by a header containing the task group identification.

Where appropriate, the header for the task control block (TCB) contains a bound unit name.

The header for each file control block (FCB) also contains the logical file number (LFN) of the file.

Work space blocks may also be labelled as FCBs, TCBs, or RBs if they appear as these structures within the same task group.

The firmware-defined fields (instruction, P-counter, I′, Z, A, R3, and B3) for each trap save area (TSA) are displayed. If the instruction is a monitor call, the function code is also displayed.

Headings for task group structures are indented to show the hierarchical relationship.

For non-system task groups, the display of the group control block (GCB) is extended to show the group's logical file table (LFT) and the logical resource table (LRT). The LFT and LRT begin at the end of the GCB.

In addition, a possible context of the remaining data and address registers (R1, R2, R4, R5, R6, R7, B1, B2, B4, B5, B6, and B7) is displayed for each trap save area. This context, which is extracted from the work space area of the trap save area, may not be valid in all cases, but, in general, is correct due to internal conventions of the MOD 400 Operating System.

DPEDIT Command

The DPEDIT command loads the Dump Edit utility program. Immediately after Dump Edit begins executing, a message is issued to the error output file giving the unique version number in the following format: DPEDIT-nnnn-mm/dd/hhmm. The message "DUMP COMPLETE" is issued to the error-out file immediately before the execution of Dump Edit terminates.

FORMAT:

DPEDIT [path] [ctl_arg]

ARGUMENT DESCRIPTIONS:

path[1]
Pathname of the memory dump file to be printed.

ctl_arg
Control arguments; zero, one, or more of the following control arguments may be entered, in any order:

{-NO_LOGICAL / -NL}
No logical dump of system control structures produced.

Default: Logical dump produced.

{-NO_PHYSICAL / -NP}
No physical dump of memory produced.

Default: Physical dump produced.

{-FROM X'address' / -FM X'address'}[2]
Low-memory address (up to four hexadecimal digits for SAF and five for LAF) of area that will appear in physical dump; must be specified in hexadecimal. This must be a real (not a virtual) address.

Default: Absolute 0.

-TO X'address'
High-memory address (up to four hexadecimal digits for SAF and five for LAF) of area that will appear in physical dump; must be specified in hexadecimal. This must be a real (not a virtual) address.

Default: High memory address of the dump file.

{-MEMORY / -MEM}[1]
Produces a dump of main memory. If both the path argument and this argument are specified, the path argument is ignored.

[1] Either the path argument or the -MEMORY control argument must be specified.

[2] If the -FROM control argument is used in conjunction with the -MEMORY control argument, then the address that is specified must be a main memory location whose virtual address is the same value as its real address.

Default: A dump is produced of the file specified in the path argument.

{-GROUP / -GP} group id [group-id] . . .

Requests the logical dump to contain task group-related information for the specified group(s) only.

Default: Task group information for all groups is included in the logical dump.

NOTE: Either the path argument or the -MEMORY control argument must be specified.

Example 1:

```
DPEDIT ^ DMPVOL>DUMPFILE -NL -TO X'3000'
```

This command loads the Dump Edit utility program and requests only a physical dump of the first 12K locations of the specified dump file.

Example 2:

```
DPEDIT -MEM
```

This command loads the Dump Edit utility program and requests a logical and physical dump of current main memory.

Example 3:

```
DPEDIT -MEM -GROUP $S $D -NP -NF
```

This command loads the Dump Edit utility program and requests a logical dump of only the System and Debugger groups from current main storage. This command suppresses display of the file management structures.

Example 4:

```
DPEDIT ^ DUMPER>DUMP256K
```

By specifying a group that does not exist, (i.e., XX) this command requests an abbreviated logical dump consisting of only the System Summary from within the specified dump file.

Operating Procedure for Dump Edit

The following steps must be performed before the Dump Edit program can be executed.

1. Mount the disk volume containing Dump Edit.
2. If Dump Edit is being used to print MDUMP output, mount the disk volume that contains the memory image obtained from the MDUMP memory dump.
3. Execute Dump Edit by specifying the DPEDIT command described previously.

DPEDIT processing can be stopped at any time by depressing the "BREAK" key. A "**BREAK**" message appears on the user's terminal when the processing stops. GCOS 6 command may be specified at this point. If the program interrupt command (PI) or the unwind command (UW) is specified, the end-of-processing details are automatically handled and control returns to the command processor with a successful subtask completion status. If the start command (SR) is specified, DPEDIT resumes processing.

When Dump Edit is used to print MDUMP output, the address mode that was in effect for MDUMP must be used for Dump Edit; i.e., the SAF version of DPEDIT processes SAF memory dumps, and the LAF version processes LAF memory dumps.

Messages

Fatal errors terminate DPEDIT processing, return control to the command processor, and post an unsuccessful subtask completion status. Fatal errors include logical I/O errors and physical I/O errors as well as DPEDIT-specific errors. Fatal error messages are written to the ERROR_OUT file by the system service, Error Handler, and are described in the *System Messages* manual. For convenience, fatal error messages that are specific to DPEDIT are summarized in Table 6-2.

TABLE 6-2. DPEDIT – SPECIFIC FATAL ERROR MESSAGES

Information	Meaning
2502 ILLEGAL NUMBER OF ARGUMENTS	The number of arguments specified in the DPEDIT command is excessive.
2503 NON-NUMERIC CHARACTER IN NUMERIC ARGUMENT	A non-numeric character was found in a DPEDIT argument where a numeric argument is required.
2507 ARGUMENT NOT RECOGNIZED	An argument has been specified in the DPEDIT command which does not conform to the defined list of control arguments.
2512 REQUIRED ARGUMENT MISSING	In certain situations a DPEDIT argument may be required. If such a situation occurs and the argument is missing, this message is produced.
2513 ADDRESS MODE INCOMPATIBILITY	The address mode (SAF or LAF) of the dump file differs from that of the executing DPEDIT utility.
2514 DUMP FILE IS INCORRECT FILE-TYPE	The dump file must be a BES-200 Relative file with no deletable records created by the CREATE_VOL (CV) utility.
2515 DUMP FILE IS INCOMPLETE	The dump file, when filled by MDUMP, did not attain a successful end-of-job condition (see Table 6-1). The dump file is therefore incomplete.

Immediately after execution of DPEDIT begins and immediately before execution terminates, a message is written to the ERROR_OUT file. These messages are explained in the description of the DPEDIT command.

Informational messages that generally reflect some condition peculiar to the data within the dump file may be interspersed with the dump information in the USER_OUT file. These messages are designed to facilitate analysis of the dump and are listed below in alphabetical order. A brief explanation of each message is included.

ADDRESS POINTER IS INVALID
A virtual address contained in the dump file is invalid.

BATCH GROUP IS ROLLED OUT
The background was rolled out when the memory dump was taken.

DATA NOT READABLE
Dump Edit tried to read the contents of a nonexistent location on the dump file, or an uncorrectable read error was encountered. If this condition occurs more than five times within a given task group, processing is terminated.

DUPLICATE FILE CONTROL BLOCK STARTING ADDRESS
The specified file control block has already been displayed.

DUPLICATE GROUP CONTROL BLOCK STARTING ADDRESS
The specified group control block has already been displayed.

DUPLICATE TASK CONTROL BLOCK STARTING ADDRESS
The specified task control block has already been displayed.

DUPLICATE WORK SPACE BLOCK

The specified work space block has already been displayed.

INPUT IS NOT A MOD400 DUMP FILE

The external dump file to be processed contains a dump of an MDT Operating System.

INSTRUCTION WHICH TRAPPED IS AN MCL AT LOCATION nnnn. FUNCTION CODE IS nnnn.

This message gives information about the associated trap save area.

INSTRUCTION: nnnn P_COUNTER"nnnn I':nnnn Z:nnnn A:nnnn R3:nnnn B:3nnnn

This message gives information about the associated trap save area.

INVALID WORK SPACE BLOCK POINTER

A work space block was encountered that does not begin on a 32-word multiple boundary.

LOGICAL DUMP CAN GO NO FURTHER. PHYSICAL DUMP IS SUGGESTED.

DPEDIT has determined that some operating system structure has been overwritten. This message can appear only during a logical dump.

NO ENTRIES

System Symbol Table is empty.

NUMBER OF ALLOCATED WORK SPACE BLOCKS EXCEEDS 60

More than 60 work space blocks have been allocated for the current group control block.

NUMBER OF BUFFER CONTROL BLOCKS EXCEEDS 25

More than 25 buffer control blocks have been allocated for the current file descriptor block.

NUMBER OF FILE CONTROL BLOCKS EXCEEDS 40

More than 40 file control blocks have been allocated for the current group control block.

NUMBER OF GROUP CONTROL BLOCKS EXCEEDS 40

More than 40 group control blocks have been allocated for the current configuration.

NUMBER OF INDIRECT REQUEST BLOCKS EXCEEDS 25

More than 25 indirect request blocks are allocated for the current task control block.

NUMBER OF TASK GROUP CONTROL BLOCKS EXCEEDS 40

More than 40 task group control blocks have been allocated for the current group control block.

NUMBER OF TRAP SAVE AREAS EXCEEDS 10

There are more than 10 trap save areas for the current task control block.

THIS WORK SPACE BLOCK IS A FILE CONTROL BLOCK

The specified work space block has appeared previously as a file control block in this task group.

THIS WORK SPACE BLOCK IS A REQUEST BLOCK

The specified work space block has appeared previously as an I/O-, a task-, or a group-request block.

THIS WORK SPACE BLOCK IS A TASK CONTROL BLOCK

The specified work space block has appeared previously as a task control block.

VIRTUAL ADDRESSING DISCONTINUITY EXISTS AT VIRTUAL ADDRESS hhhh. DUMP WILL RESUME AT VIRTUAL ADDRESS kkkk.

The virtual address, hhhh, is invalid. The physical dump from main memory will attempt to restart at virtual address, kkkk.

VIRTUAL ADDRESSING ERROR . . . INVALID OFFSET

A virtual address has been encountered whose offset field exceeds the size field of its virtual space segment descriptor.

VIRTUAL ADDRESSING ERROR . . . INVALID SEGMENT

A virtual address has been encountered whose segment field designates an invalid virtual space segment descriptor.

APPENDIX A
INTERPRETING AND USING MEMORY DUMPS

Memory dumps can be obtained by using Debug or Dump Edit. It is preferable to use dumps produced by Dump Edit; they are in edited format and are much easier to interpret (see Section 6).

This appendix describes significant locations on memory dumps, how to interpret the contents of locations on memory dumps, and how to use memory dumps to perform the following procedures:

- o Finding the location in memory of your code
- o Determining where a trap occurred
- o Determining the state of execution of your code

A trap is a special software- or hardware-related condition that may occur during the execution of a task. Many traps are caused by an error, but a few, such as the Monitor Call, are not. The above procedures may have to be performed if a trap message is issued. Traps and trap messages are described in detail in the "Trap Handling" section of the *System Services Macro Calls* manual.

NOTE: In this appendix, all references to memory locations and offsets are for both SAF and LAF modes (short-address form and long-address form, respectively), and offsets always are in hexadecimal. LAF address and offsets are enclosed within parentheses and indicate the two-word form.

SIGNIFICANT LOCATIONS ON MEMORY DUMPS

Table A-1 describes memory locations on the dump that it may be useful to refer to during debugging. It is assumed that you are familiar with the data structures referenced. Brief definitions of these data structures are contained in the glossary of the *System Concepts* manual. Figure A-1 illustrates a map of systems data structures.

TABLE A-1. SIGNIFICANT LOCATIONS ON MEMORY DUMP

Memory Address	Meaning
0010 (0010/0011)	Head of queue of available trap save areas (TSA's).
0018 (0018/0019)	Pointer to system control block (SCB). This is the key to locating all system data structures.
0020-0023	Level activity flags for levels 0 through 63. Bits ON indicate which levels are ready to execute; the lowest of these levels is the level currently executing (i.e., the active level). The level 63 bit always is on. The clock level bit (4) may be on, and the Debug level bit is on if the dump resulted from a Debug DP directive.
0052-007F (0024-007F)	Trap vectors. Each trap vector is associated with a specific trap condition and points to that trap handler's entry address. The trap vector for trap number 1 is in location 007F (7E/7F). The trap vectors for subsequent trap numbers are in descending, contiguous, locations; i.e., the trap vector for trap number 2 is in location 007E (7C/7D).
0080-00BF (0080-00FF)	Pointers to interrupt save areas (ISA's) for levels 0 through 63, respectively. A null value means there is no dedicated task (i.e., a driver) or nondedicated task ready to execute on the specified level.

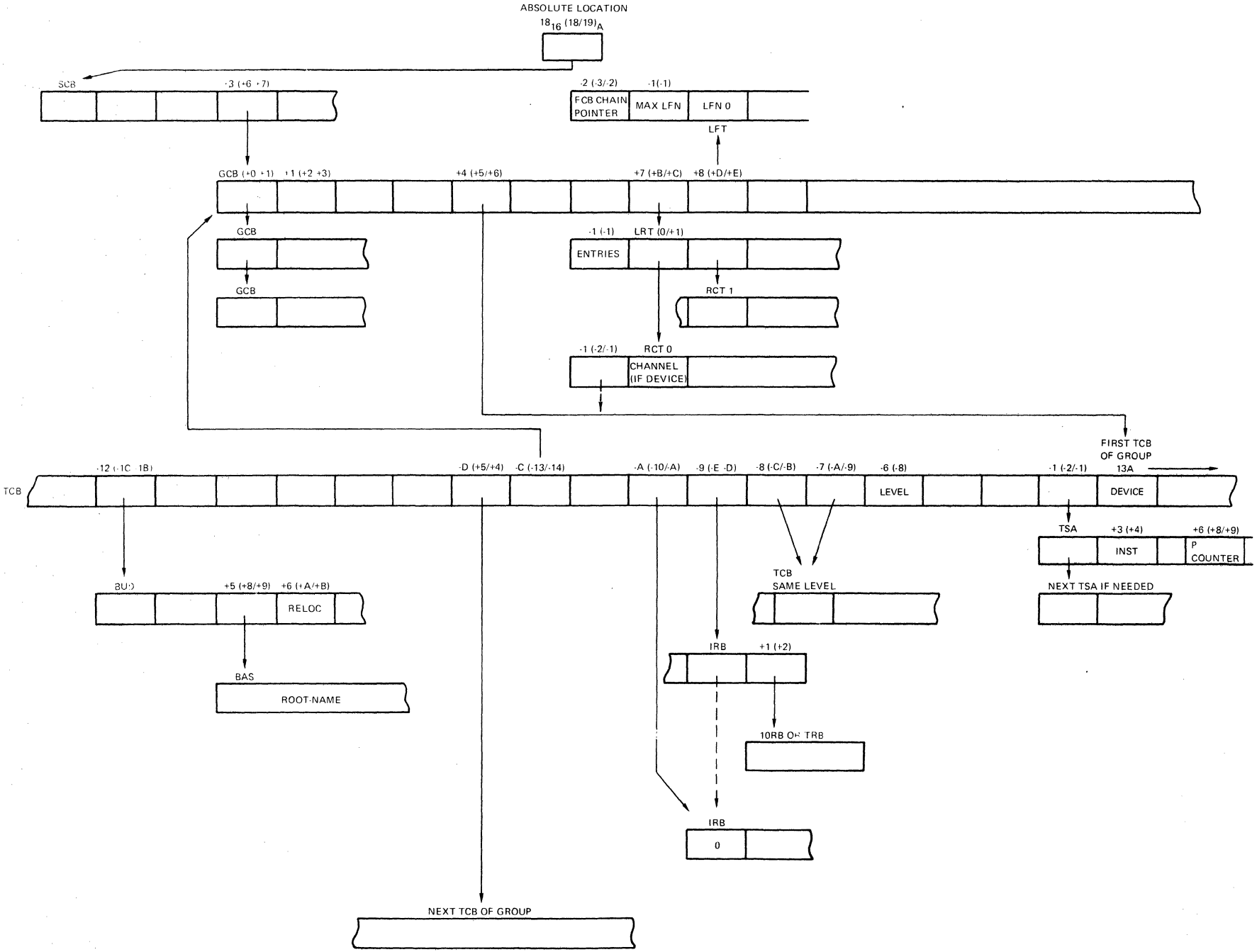


Figure A-1. Data Structure Map

Locations Relative to the System Control Block or Group Control Block

SCB+3 (+6)
Pointer to first group control block (GCB)

GCB+0 (+0/+1)
Pointer to next GCB in linked list of GCB's.

GCB+1 (+2)
Task group identification ($S is the system group; $B is the batch group). The system will convert your user identification to non-ASCII representation.

GCB+8 (+D/+E)
Pointer to LFN0 of logical file table (LFT).

GCB+7 (+B/+C)
Pointer to LRN0 of task group's logical resource table (LRT).

GCB+4 (+5/+6)
Pointer to first task control block (TCB) of the group.

LRT-1 (-1)
Number of entries in the LRT.

LRT+0 (+0/+1)
Pointer to LRN 0's resource control table (RCT); the RCT's for subsequent LRN's are in contiguous, ascending locations (LRT+1 points to LRN 1's RCT). A null entry indicates that the associated LRN is not used.

NOTE: Within an RCT, location 0 is the channel number of the resource if it is an input/output device.

RCT-1 (-2/-1)
Pointer to task control block (TCB) for that resource.

Locations Relative to the Task Control Block (TCB) Pointer for the Desired Priority Level

TCB-6 (-8)
Hardware-assigned priority level of the task.

TCB-12 (-1C/-1B)
Pointer to current bound unit BUD.

TCB-A (-10/-A)
Pointer to end of queue of requests for the task.

TCB-9 (-E/-D)
Pointer to start of queue of requests for the task (e.g., I/O requests for a driver).

TCB-C (-14/-13)
Pointer to the group control block (GCB) for the group to which this task belongs.

TCB-D (-15/-14)
Link to the queue of this group's TCB's.

TCB-7 (-A/-9)
Pointer to last TCB on that priority level.

TCB-8 (-C/-B)
Link to other task control blocks (TCB's) of the same or different task groups assigned to the same level.

TCB-1 (-2/-1)
Pointer to the queue of trap save areas (TSA's) for the task. (Trap save areas are described in detail in the "Trap Handling" section of the *System Service Macro Calls* manual.) If a TSA is present, the task is executing system code or a user trap; if no TSA is present, check the program counter in the interrupt save area (ISA) portion of the TCB to determine the tasks's progress.

TCB+0
Device word, including channel number and level number. This entry is null if the task does not drive a device.

TCB+n
Hardware ISA.

INTERPRETING THE CONTENTS OF A DPEDIT LOGICAL DUMP

This section addresses dump interpretation when the DPEDIT dump format is used.

Finding the Location in Memory of Your Code

Locate your group-id and the TCB for your bound unit (BU). The first six characters of the BU filename are printed beside each TCB of the group.

The address at TCB-11(-1B/1A) is the start address of the BU. Calculate relative zero of the BU by subtracting the relative start address on its link map from this address.

Determining the State of Execution of Your Code at the Time of the Dump

Dump analysis begins with gathering all relevant information: the dump itself, the console hard-copy (if any) of the activity of a particular group (or groups), copies of the CLM_USER and >START_UP.E files, plus any link maps.

These materials are required to understand the environment of the system represented in the dump.

Three conditions are discussed below:

1. Halt at level 2
2. User level active at the time of dump
3. No level active at the time of dump, except level 63.

Halt at Level 2

Examination of the level activity indicators at locations 20-23 confirms that level 2 is active. The system will force this condition to occur if either TSA or IRB resources are exhausted (see CLM SYS directive). Note that once level 2 becomes active, other lesser priority levels may activate but will not receive CPU time and should be ignored.

The D1 register contains an ASCII IR (4952) when IRB exhaustion has occurred. Location 10 (10/11) is zero when TSA exhaustion has occurred.

If this symptom persists after augmenting the number of TSA/IRBs available to the system, it is possible that either your code or the system is improperly altering the TSA/IRB chains. To verify this, take a memory dump immediately after system startup. This allows easy location of the TSA chains from location 10 (10/11) and the IRB chains from the first location of the SCB. Compare this dump to one taken after all TSA/IRBs are supposedly exhausted to verify that they really are. If the system is suspect, supply both dumps to Honeywell if you have a maintenance contract. TSAs can also be exhausted by a recursive trap. A recursive trap uses up all available TSAs. Adding TSAs simply allows for greater recursion. In this instance, the system is suspect and dumps should be supplied to Honeywell.

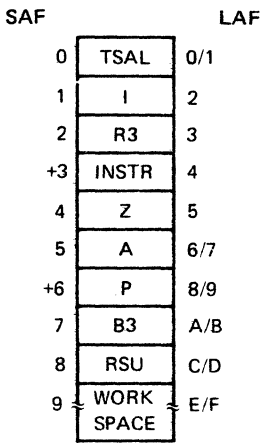
User Level Active at the Time of Dump

This often indicates a halt or software loop condition on the active level. When a level is active, the pointer to the TCB associated with the code running is in the interrupt vector for that level. Match the TCB pointer with the TCBs listed for the groups present in the system. When a level is active, use the P-counter in the ISA portion of the TCB to locate the software running at the last time this level's context was saved. Since the system clock is active on level 4, the P-counter in the ISA for this level is usually helpful. It is also helpful to record the contents of R/B registers and EO when entering STEP mode at the control panel prior to taking the dump.

No Level Active at the Time of Dump, Except for Level 63

This condition usually indicates a system failure in that all tasks have been suspended and none are being reactivated. In this situation it is helpful to determine the conditions existing at this time. To do this, examine all TCBs in groups other than $S group. If the TCB under examination has not experienced a default trap condition, it may or may not have an associated TSA. If a TSA is shown, DPEDIT will display the monitor call function code if the trapped instruction is 0001 (monitor call generic). The function may be decoded using the numerical listing included in this appendix.

When the system is called for a monitor function, only those registers that must be preserved by the system are saved in the TSA workspace. The saved registers are: B7, B6, B5, B1, R5, R4, M1, beginning at TSA location +9 (+E//F). The trap save area (TSA) is illustrated below:



DETERMINING WHERE A TRAP PROCESSED BY THE SYSTEM DEFAULT HANDLER OCCURRED IN YOUR CODE

If a trap message occurs on the operator terminal from the system default trap handler; i.e., (id) BUname (0303zz) level, the TCB of the referenced task group may be located using the bound unit name (BUname). In this situation, unless the TCB is subsequently re-requested, the last two areas associated with the TCB are related to the system handling of the trap. The first TSA following the TCB was used by the system to forceably terminate the task request in progress when the trap occurred. Your information is found in the next TSA associated with the TCB. It contains the hardware information described in the previous section of this appendix, followed by a complete set of registers current when the trap occurred. The order of the registers, beginning at location +9 (+E/F) of the TSA, is: B7, B6, B5, B4, B2, B1, I, R7, R6, R5, R4, R2, R1, M1 (B3, R3, I are already in the TSA). When the TCB has been re-requested, only this second TSA remains attached to the TCB.

FINDING THE LOCATION IN MEMORY OF YOUR CODE

The three activities above may be performed without aid of the DPEDIT logical dump presentation. The examination of TCB contents is the same once the TCB is located. Use the following procedure to find the TCBs for your group.

1. Go to location 0018 (18/19); this location contains a pointer to the system control block (SCB).
2. Go to location SCB+3 (+6); this location contains a pointer to the first group control block (GCB); the first word links to other GCB's in the system. Determine the group id at GCB+1 (+2/+3).
3. Go to location GCB+4 (+5/+6) to determine the location of the first task control block (TCB) of the task group.
4. Go to location TCB-12 (-1D/-1C) to determine the location of your current bound unit descriptor (BUD).
5. Go to location BUD+6 (+A/+B). This location is the relocation factor of the bound unit; your code should start at this location.
6. To confirm that your code does start at location BUD+6 (+A/+B), go to location BUD+5 (+8/+9); this location points to the location of the bound unit attribute section (BAS).
7. Go to location BAS+0 to determine the bound unit's root name; this name should be the same file name (i.e., the same leading six characters) that you specified in the name argument of the LINKER command.
8. If you did not find the root name for which you were looking, go to location TCB-D (-16/-15); this location points to the next TCB of the task group. Follow through the chain of TCB's until you find your task's task control block.

INTERPRETING THE MONITOR CALL NUMBER ON MEMORY DUMPS

Table A-2 is ordered numerically to facilitate identification of a monitor call function code, and provides a brief description of each Executive monitor call.

TABLE A-2. SUMMARY OF EXECUTIVE MONITOR CALLS

Monitor Call Number	Function Description	Macro Call Name
0100	Wait for operation complete	$WAIT
0101	Wait on request list	$WAITL
0102	Test completion status	$TEST
0103	Terminate request start address not modified	$TRMRQ
0104	Terminate request $B4 has new start address	$TRMRQ
0105	Dequeue IRB	
0106	Post IRB	
0107	Return request block address	$RBADD
0108	Locate user RCT	
0200	Request I/O transfer	$RQIO
0202	Disable device	$DSDV
0203	Reset device attention	$RDVAT
0204	Enable device	$ENDV
0205	Start error logging	$ELST
0207	Exchange error log	$ELEX
0208	Get error logging information for this device	$ELCT
0209	End error logging	$ELEND
0402	Get memory	$GMEM
0403	Get available memory	$GMEM
0404	Return memory	$RMEM
0405	Return partial block of memory	$RMEM
0406	Status memory pool	$STMP
0500	Request clock	$RQCL
0501	Cancel clock request	$CNCRQ
0502	Suspend for interval	$SUSPN
0503	Suspend until time	$SUSPN
0504	External date/time - convert to	$EXTDT
0505	External time - convert to	$EXTIM
0506	Get date/time	$GDTM
0507	Internal date/time - convert to	$INDTM
0508	Set system date/time	
0600	Request semaphore	$RQSM
0601	Cancel semaphore request	$CNSRQ
0602	Reserve resource	$RSVSM
0603	Release semaphore	$RLSM
0604	Define semaphore	$DESM
0700	Execute overlay	$OVEXC
0701	Load overlay	$OVLD
0703	Status overlay	$OVST
0705	Reserve area and execute overlay	$OVRSV
0706	Release overlay area	$OVRSL
0707	Release, wait on RB and recall	$OVRCL

TABLE A-2 (CONT). SUMMARY OF EXECUTIVE MONITOR CALLS

Monitor Call Number	Function Description	Macro Call Name
070A	Create overlay area	$CROAT
0700	Unload overlay	$OVUN
0800	User input file - read	$USIN
0801	User output file - write	$USOUT
0802	Command infile (read command-in file)	$CIN
0803	Error output file - write to	$EROUT
0804	New user input file - redefine	$NUIN
0805	New user output file - redefine	$NUOUT
0806	New command input - reset	
0900	Operator information message - display	$OPMSG
0901	Operator response message - display	$OPRSP
0A00	Trap handler connect	$TRPHD
0902	Console message suppression - on	$CMSUP
0903	Console message suppression - off	$CMSUP
0A01	Enable user trap	$ENTRP
0A02	Disable user trap	$DSTRP
0A04	Trap handler query	$TRPHD
0B00	Read external switches	$RDSW
0B01	Set external switches	$SETSW
0B02	Clear external switches	$CLRSW
0C00	Request task	$RQTSK
0C02	Create task; same bound unit as issuing	$CRTSK
0C03	Create task; different bound unit than issuing	$CRTSK
0C04	Delete task	$DLTSK
0C05	Spawn task; same bound unit as issuing	$SPTSK
0C06	Spawn task; different bound unit than issuing	$SPTSK
0C08	Command line - process synchronously	$CMDLN
0D00	Request group	$RQGRP
0D03	Create group	$CRGRP
0D04	Delete group	$DLGRP
0D05	Spawn group	$SPGRP
0D07	Abort group request	$ABGRQ
0D08	Suspend group	$SUSPG
0D09	Activate group	$ACTVG
0D0A	Abort group	$ABGRP
0D0B	New process	$NPROC
0E00	Request batch execution	$RQBAT
0F00	Report error condition	$RPTER
0F01	Report error condition	$RPTER
1010	Associate file	$ASFIL
1015	Disassociate file	$DSFIL

TABLE A-2 (CONT). SUMMARY OF EXECUTIVE MONITOR CALLS

Monitor Call Number	Function Description	Macro Call Name
1020	Get file	$GTFIL
1025	Remove file	$RMFIL
1030	Create file	$CRFIL
1035	Release file	$RLFIL
1040	Rename file/directory	$RNFIL
1050	Open file (preserve)	$OPFIL
1051	Open file (renew)	$OPFIL
1055	Close file (normal)	$CLFIL
1056	Close file (leave)	$CLFIL
1057	Close file (unload)	$CLFIL
1060	Get file information	$GIFIL
1061	Test file I/O	$TSFIL
1062	Test file for input	$TIFIL
1063	Test file for output	$TOFIL
1064	Wait for file input	$WIFIL
1065	Wait for file output	$WOFIL
10A0	Create directory	$CRDIR
10A5	Release directory	$RLDIR
10B0	Change working directory	$CWDIR
10C0	Get working directory	$GWDIR
10D0	Expand pathname	$XPATH
1110	Read record	$RDREC
1111	Read record (with key)	$RDREC
1112	Read record (position = key)	$RDREC
1113	Read record (position > key)	$RDREC
1114	Read record (position ≥ key)	$RDREC
1115	Read record (position forward)	$RDREC
1116	Read record (position backward)	$RDREC
1120	Write record	$WRREC
1121	Write record (with key)	$WRREC
1122	Write record (position = key)	$WRREC
1123	Write record (position > key)	$WRREC
1124	Write record (position ≥ key)	$WRREC
1125	Write record (position forward)	$WRREC
1126	Write record (position backward)	$WRREC
1130	Delete record	$DLREC
1131	Delete record (with key)	$DLREC
1140	Rewrite record	$RWREC
1141	Rewrite record (with key)	$RWREC
1150	Unlock record	$ULREC
1200	Read block (normal)	$RDBLK
1201	Read block (position to tape mark)	$RDBLK

TABLE A-2 (CONT). SUMMARY OF EXECUTIVE MONITOR CALLS

Monitor Call Number	Function Description	Macro Call Name
1202	Read block (position to beginning of tape)	$RDBLK
1203	Read block (position on blocks)	$RDBLK
1204	Read block (position to end of tape)	$RDBLK
1210	Write block (normal)	$WRBLK
1211	Write block (write to tape mark)	$WRBLK
1220	Wait block	$WTBLK
130A	Set terminal characteristics	$STTY
1400	User identification	$USRID
1401	Task group person identification	$PERID
1402	Account identifier	$ACTID
1403	Task group mode identification	$MODID
1404	System identification	$SYSID
1406	Bound unit name	$BUID
140B	Home directory name	$HDIR
140C	Task group input file name	$TGIN
1501	Accept message group	$MACPT
1502	Initiate message group	$MINIT
1503	Receive	$MRECV
1504	Terminate message group	$MTMG
1505	Send	$MSEND
1506	Cancel message enclosure	$MCME
1507	Count message group	$MCMG
1509	Wait on message group	$MWAIT
1702	Cancel request for terminal	$CANRQ
1703	Request terminal	$RQTML
1704	Release terminal	$RLTML
1B00	Set dial	$SDL
---	Clock request block template - create	$CRB
---	Clock request block template - offsets	$CRBD
---	Create file parameter block structure - offsets	$CRPSB
---	File information block - create	$FIB
---	Get file information file attributes block - offsets	$GIFAB
---	Get file information, key descriptors block - offsets	$GIKDB
---	Get file information, parameter structure block - offsets	$GIPSB
---	Get file, parameter structure block - offsets	$GTPSB
---	Input/Output request block template - create	$IORB
---	Input/Output request block template - offsets	$IORBD

TABLE A-2 (CONT). SUMMARY OF EXECUTIVE MONITOR CALLS

Monitor Call Number	Function Description	Macro Call Name
---	Parameter structure block - generate	$PRBLK
---	Request block template	$RBD
---	Return sequence - establish	$RETRN
---	Semaphore request block - create	$SRB
---	Semaphore request block template - offsets	$SRBD
---	File information block - offsets	$TFIB
---	Task request block - create	$TRB
---	Template task request block - offsets	$TRBD
---	Wait list - generate	$WLIST

CUT ALONG LINE

HONEYWELL INFORMATION SYSTEMS

Technical Publications Remarks Form

TITLE	SERIES 60 (LEVEL 6) GCOS 6 MOD 400 PROGRAM EXECUTION AND CHECKOUT ADDENDUM A	ORDER NO.	CB21A, REV. 0
		DATED	JUNE 1978

ERRORS IN PUBLICATION

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Your comments will be promptly investigated by appropriate technical personnel and action will be taken as required. If you require a written reply, check here and furnish complete mailing address below. ☐

FROM: NAME _____ DATE _____

TITLE _____

COMPANY _____

ADDRESS _____

_____

CUT ALONG LINE

PLEASE FOLD AND TAPE –
NOTE: U. S. Postal Service will not deliver stapled forms

FOLD ALONG LINE

FIRST CLASS
PERMIT NO. 39531
WALTHAM, MA
02154

Business Reply Mail
Postage Stamp Not Necessary if Mailed in the United States

Postage Will Be Paid By:

HONEYWELL INFORMATION SYSTEMS
200 SMITH STREET
WALTHAM, MA 02154

ATTENTION: PUBLICATIONS, MS 486

FOLD ALONG LINE

Honeywell