Honeywell

LEVEL 6

SOFTWARE

GCOS 6 MOD 400 PROGRAM EXECUTION AND CHECKOUT

SERIES 60 (LEVEL 6)

GCOS 6 MOD 400 PROGRAM EXECUTION AND CHECKOUT MANUAL

SUBJECT

Detailed Description of Series 60 (Level 6) GCOS Program Execution and Checkout Procedures

SOFTWARE SUPPORTED

This publication supports Release 0100 of the Series 60 (Level 6) MOD 400 Operating System; see the Manual Directory of the latest *GCOS 6 MOD 400 System Concepts* manual (Order No. CB20) for information as to later release supported by this manual.

ORDER NUMBER

CB21, Rev. 0

November 1977

Honeywell

PREFACE

This manual describes the GCOS 6 MOD 400 program execution and checkout procedures. Unless stated otherwise herein, the term GCOS refers to the GCOS 6 MOD 400 software; the term Level 6 refers to the Series 60 (Level 6) hardware on which the software executes.

Section 1 summarizes how to access files via pathnames, and describes in detail the suffixes that are appended to file names. It is important that you understand these concepts before proceeding with the manual.

Section 2 describes how to load the Linker, Linker functions, and directives that control execution of the Linker.

Section 3 provides a summary of the procedures and commands used in program execution.

Section 4 describes how to load Patch, and includes detailed descriptions of Patch directives.

Section 5 describes how to debug programs using Debug and other methods.

Section 6 describes how to load and use the MDUMP and Dump Edit utility programs.

Appendix A describes, in detail, how to interpret memory dumps. This appendix includes procedures for determining where a trap occurred, finding the location in memory of your code, and determining where execution of your code terminated.

File No.: 1S23

CB21

MANUAL DIRECTORY

The following publications comprise the GCOS 6 manual set. The Manual Directory in the latest GCOS 6 MOD 400 System Concepts manual 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*
CB30	*Remote Batch Facility User's Guide*
CB31	*Data Entry Facility User's Guide*
CB33	*Level 6/Level 6 File Transmission*
CB34	*Level 6/Level 62 File Transmission*
CB35	*Level 6/Level 64 (Release 0300) File Transmission*
CB36	*Level 6/Level 66 File Transmission*
CB37	*Level 6/Series 200/2000 File Transmission*
CB38	*Level 6/BSC 2780 File Transmission*
CB39	*Level 6/Level 64 (Release 0220) File Transmission*

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*

CONTENTS

ILLUSTRATIONS

TABLES

SECTION 1

OVERVIEW OF PROGRAM EXECUTION AND CHECKOUT

Honeywell supplies the necessary tasks to create a source unit and convert it into an executable format (including error detection and correction) or to apply a patch. These tasks are described in subsequent sections of this manual and in the *Program Preparation* manual.

Program checkout can be performed after you first perform both the initial system startup procedure and a specialized system startup procedure. The initial and specialized startup procedures are described in the "Startup and Configuration" section of the *System Building* manual. The equipment required for program preparation is described in the "Equipment Requirements" section of the *Systems Concepts* manual.

Program checkout is described below and illustrated in Figure 1-1. A source file is edited, then compiled or assembled as described in the *Program Preparation* manual. Before execution, separately assembled and/or compiled object units must be linked by the Linker to form a bound unit. A bound unit comprises a root, or a root and one or more overlays. A root is the portion of a bound unit that is loaded into memory when the Loader is requested to load a bound unit.[1] The root remains in memory as long as there are tasks executing on its behalf, unless the LDBU configuration directive 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.

After linking and loading the bound unit, you can control execution of a program and make desired changes while the program is executing by using Debug. Breakpoints can be set to determine which code is executing, and specified registers and memory locations can be displayed and, if desired, changed. If there is not enough memory for Debug, you can perform debugging by using Patch to append monitor points. Patch permits you to add patches to and/or delete patches from object units and bound units.

There are three methods of obtaining memory dumps. While a program is executing, you can obtain a memory dump by using either Debug or the Dump Edit utility program; dumps produced by Dump Edit are in edited format and are much easier to interpret. If an executing program encounters a problem and it aborts or a halt occurs, to obtain a memory dump you may use just Dump Edit or you may first dump memory to a disk file by using the MDUMP utility program and then print the memory dump by using Dump Edit. To dump the contents of all or part of the Multiline Communications Processor memory, you can use the DUMCP dump routine, which is described in the *Communications Processing* manual.

NOTES

1. If you are going to perform program checkout while simultaneously executing other online tasks in the foreground, you must be familiar with the *System Concepts* manual.
2. Throughout this manual there are references to the create group, enter group request, spawn group, and enter batch request commands; these commands are described in the *Commands* manual.

[1] The root is loaded when an –EFN argument is specified in a create group or spawn group command, or an LDBU configuration directive is specified (see the *System Building* manual).

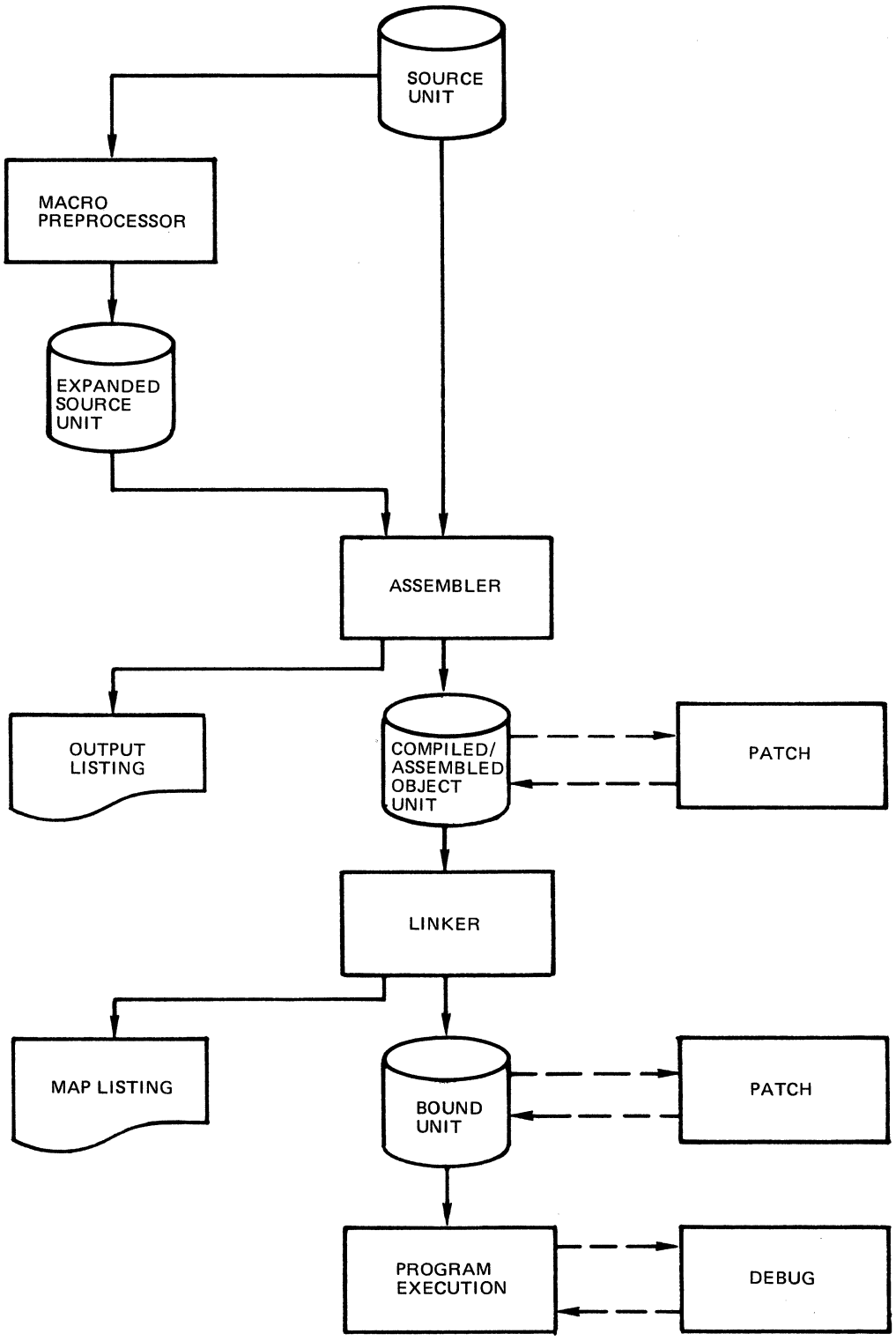
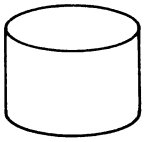


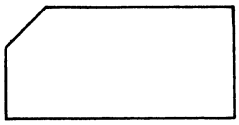
Figure 1-1. Program Execution and Checkout Procedures

SYMBOLS USED IN THIS MANUAL

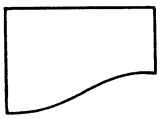
Processing; indicates any kind of processing function.



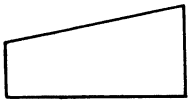
Online storage of information; e.g., diskette or cartridge disk.



Input from card reader.



Document; e.g., printer output.



Manual input; i.e., operator's terminal or another terminal.

Mandatory; indicates that the designated flow of information, type of processing, input, or output is required.

UPPERCASE CHARACTERS	Reserved words or symbols, must be entered or used exactly as shown.
lower case characters	Symbolic name or value; you must supply the exact value.
brackets [ ]	Optional information.
braces { }	An enclosed entry must be selected.
ellipses ...	There may be multiple entries of the immediately preceding type of information.

SECTION 2
LINKER

The Linker combines separately assembled and/or compiled object units, which can also be called compile 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 (4 byte-address) or Long Address Form (LAF) mode (8 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).

Each bound unit to be linked requires a separate 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

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

group 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

The Linker requires that each of its input file names contain a .O suffix. When you specify a file name in a link directive, or in the LINKER command name of a file that will contain the bound unit, suffixes are omitted, the Linker will not append a suffix to the bound unit name. 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 or default base name.

It is important to note that the Linker appends suffixes to specified file names.

Table 2-1 summarizes how file names are designated.

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 argument.

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.

Resolving External References

The Linker resolves the addresses or values of external symbol references it finds in object units being linked. The references can be between object units comprising the root, between object units comprising an overlay, between overlays, or between the root and an overlay. A symbol can be defined in one bound unit and referenced in another bound unit. (The symbol must exist in the system symbol table, as the result of an LBDU configuration directive performed for the module in which the symbol is defined. Refer to the *System Building* manual.)

Creating a Symbol Table

A symbol table is a data structure created by the Linker for resolving external references. When the Linker encounters external references to symbols or definitions of symbols, it creates an entry in the symbol table. When that entry is defined, the Linker updates the symbol's entry in the symbol table and all references to the symbol. A symbol can be defined within an object unit, or by an LDEF or VDEF directive. (LDEF and VDEF are explained later in this section under "Linker Directive Descriptions.") A list of defined and/or undefined symbols can be obtained by producing a link map.

Producing a Link Map

A link map is a listing of information in the symbol table. A symbol can be defined in an object unit as a value or location, in the LDEF directive as a location, or in the VDEF directive as a value. If a symbol is defined as a location, the map contains the symbol name and its relative address. If a symbol is defined as a value, the map contains the symbol name and its value. The map also lists the name of each undefined symbol and the relative address of the latest reference to it.

A link map can be produced at any time during the linking process by specifying the MAP or MAPU directive. It is written to the file name .M in the working directory, unless the -COUT argument was specified in the LINKER command. (-COUT permits you to assign the list file to disk, a printer, the operator's terminal, or another terminal.) If maps are assigned to disk, a disk file with variable length records is created; the first character of each record is a print control character.

FUNCTIONAL GROUPS OF LINKER DIRECTIVES

The general functions of Linker directives are listed and described below. For more detailed information, see "Linker Directive Descriptions" later in this section.

- o Specify object unit(s) to be linked
- o Specify location(s) of object unit(s) to be linked
- o Create root and optional overlay(s)
- o Produce link map(s)
- o Define external symbols
- o Protect or purge symbols
- o Designate that the last Linker directive has been entered

Specifying Object Unit(s) to be Linked

Directives:
LINK
LINKN
LINKO

LINK, LINKN and LINKO designate that one or more specified object units are to be linked. Object units specified in LINK directives are not linked immediately; their names are put into a link request list. Once a directive has been entered which requires that all

preceding link requests are honored, linking begins. Specified object units in the primary input directory are linked before specified object units in the secondary input directory; within each directory, the object units are linked in the order in which they were requested.

LINKN causes the Linker to link object units already named in the link request list, and then to link object units specified in the LINKN directive, in the order in which they were requested.

LINKO is essentially the same as LINKN, except that all embedded directives in the named object unit(s) are ignored by the Linker.

The order in which object units are linked may be important if overlays exist.

NOTE: The Linker appends the suffix .O to each specified object unit name; when the Linker searches for an object unit name, it searches for the name including the suffix.

Specifying Location(s) of Object Unit(s) to be Linked

Directives:
- IN
- LIB
- LIB2
- LIB3
- LIB4
- LSR

Object units to be linked must be in at least one directory. The primary directory is the first directory searched by the Linker; the secondary directory, if there is one, is the second (last) directory searched; and so on. When the Linker is loaded into memory, the primary directory is the working directory, and there are no other directories.

IN permits you to designate a different directory as the primary directory.

LIB designates a directory as the secondary directory

LIB2 designates the third directory to be searched.

LIB3 designates the fourth directory to be searched.

LIB4 designates the fifth directory to be searched.

LSR produces a list of the directories in the order they are to be searched.

Each of these directives may be specified any number of times.

Creating a Root and Optional Overlay(s)

Directives:
- BASE
- START
- IST
- SHARE
- SYS
- LINK
- LINKN
- LINKO
- OVLY
- FLOVLY
- CC
- QUIT

The BASE directive defines, for subsequent object units to be linked, the relative load address within the bound unit.

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. Required.

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.

-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:

-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).

Example:

```
LINKER MYPROG -IN^MYDISK>CNL -COUT >SPD>LPT00 -SIZE 06
```

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.
{ROOT, OVLY}	A SLIC bound unit may be executed in either SAF or LAF mode.
BASE	Name of the root or overlay.
ST	Base address of the root or overlay.
SFUI	Start address of the root or overlay.
	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 parameter 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:

```
        {$                                  }
        {%                                  }
        {X'address'                         }
BASE    {=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.

$
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 7FFF (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 TEXT -COUT >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.

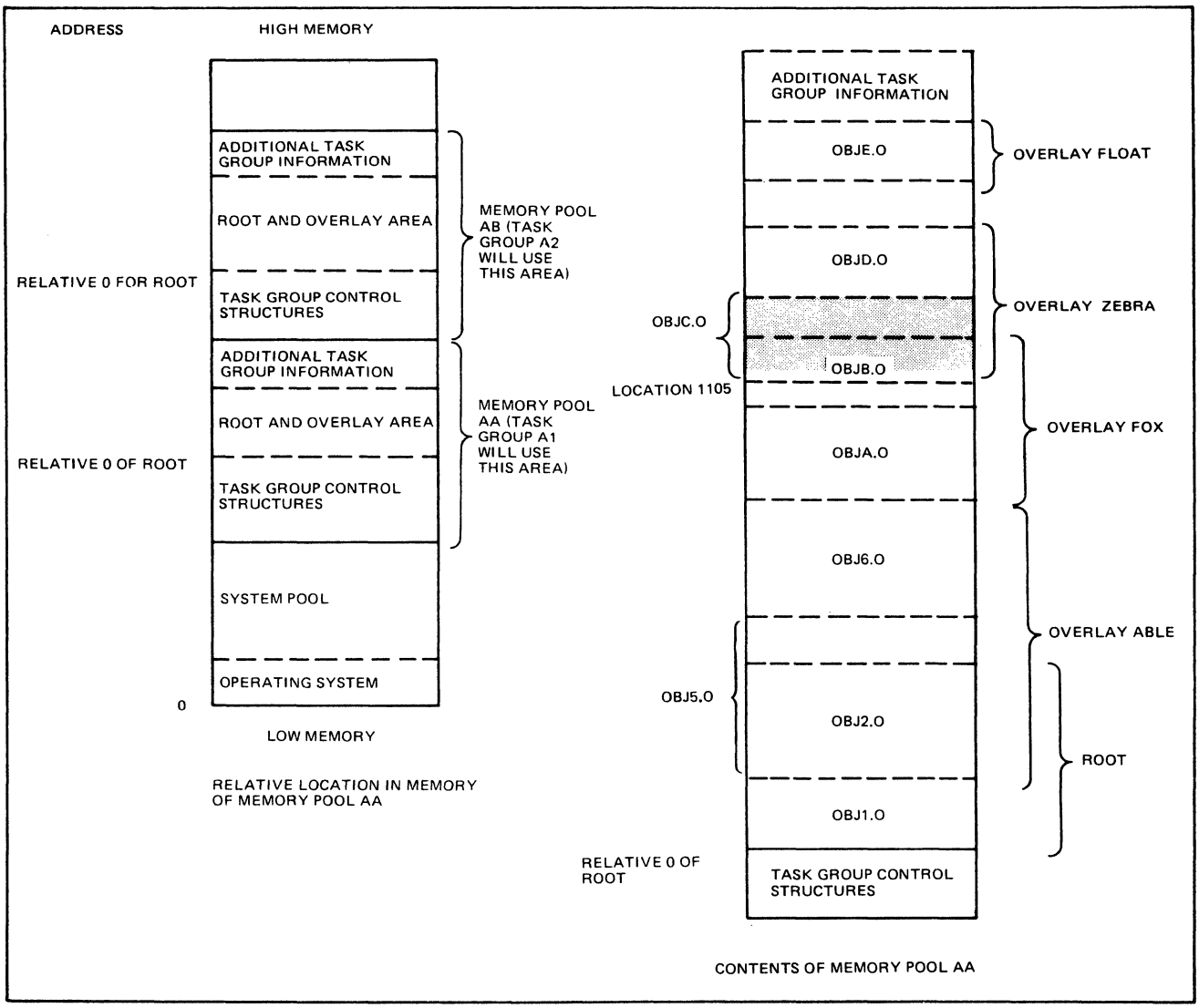


Figure 2-1. Schematic of Previous Example Illustrating Usage of BASE Directives

BASE	
BASE =OBJ2	Subsequent object unit(s) constituting overlay ABLE will be linked starting at the base address of the object unit OBJ2.O; this address can be determined from the map. Unprotected symbols that define locations equal to or greater than the address of OBJ2 are removed from the symbol table.
LINK OBJ5	Requests that OBJ5.O be linked.
MAP	
LINK OBJ6	Requests that OBJ6.O be linked.
OVLY FOX	Designates the end of the above overlay, and specifies that a nonfloatable overlay named FOX immediately follows. The Linker assigns the number 01 to this overlay.
BASE $	Subsequent object unit(s) constituting the overlay named FOX will be linked starting at one location higher than the ending address of OBJ6.O. This is the default BASE address, so BASE $ need not be specified.
LINK OBJA	Requests that OBJA.O be linked.
LINK OBJB	Requests that OBJB.O be linked.
MAP	
OVLY ZEBRA	Designates end of above overlay 01 and names subsequent nonfloatable overlay. The Linker assigns the number 02 to this overlay.
BASE X'1105'	Designates that subsequent object units constituting overlay ZEBRA will be linked starting at relative location 1105.
LINK OBJC	Object unit OBJC.O will be linked starting at relative location 1105.
LINK OBJD	Requests that OBJD.O be linked.
MAP	
FLOVLY FLOAT	Designates end of above overlay, and that a floatable overlay named FLOAT immediately follows. The Linker assigns the number 03 to this overlay. This overlay will be linked starting at the default base address of 0.
LINK OBJE	Requests that OBJE.O be linked.
MAP	
QUIT	

Call-Cancel Directive (CC)

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 by COBOL; 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, as originally specified in the COMM directive, which is to be protected.

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.

FLOVLY/IN

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 must enter the IN directive and specify in that directive the working directory's 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.)

Example 1:

```
IN^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 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.
IN^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 code that you want to execute only once immediately after the root is loaded. After initialization code is executed, the space is made available for overlays.

The external symbol must be specified in an EDEF directive.

FORMAT:

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

ARGUMENT DESCRIPTION:

external symbol
Symbol defined within the *root* as an external location.

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 7FFF (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.

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.

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.)

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 $\left\{\begin{matrix}2\\3\\4\end{matrix}\right\}$ 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:

$$\text{LIB}\left\{\begin{matrix}2\\3\\4\end{matrix}\right\}\Delta\text{path}$$

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.

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 must be one to six alphanumeric characters and must not include a suffix; the first character must be a letter or a dollar sign ($). The Linker will search for the specified object unit name, with a .O suffix.

Example 1:

```
LINK FIRST
```

This directive causes the Linker to link the object unit names 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.

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.

Example 2:

This example illustrates the order in which object units are linked if LINKN directives are used in conjunction with LINK directives, and there are embedded LINKN directives.

LINK A	Requests the linking of object unit A.O; this name is put into the link request list.
LINK B	Requests the linking of object unit B.O; this name is put into the link request list. In this example, B.O contains an embedded LINK directive to link object unit C.O.
LINKN D,G	Requests the linking of object units D.O and G.O.

In this example, all of the specified object units are in the primary directory, and D.O contains an embedded LINK directive to link object unit E.O.

When the LINKN directive is executed, the Linker will link the object units in the following order:

A.O (requested in first LINK directive)
B.O (requested in second LINK directive)
C.O (requested in a LINK directive embedded in object unit B.O)
D.O (first object unit requested in LINKN)
E.O (requested in an embedded directive in object unit D.O)
G.O (second object unit requested in LINKN)

LINKO Directive

The LINKO directive is essentially the same as the LINKN directive, except that all embedded link directives in the named object units are ignored.

Only the object units named are linked.

FORMAT:

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

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 .0 to each object unit name, and searches for the specified object unit name including the suffix.

LSR Directive

The LSR directive lists the Linker search rules; that is, the directories to be searched by the Linker for the object unit(s) are listed in the order in which they will be searched.

FORMAT:

LSR

MAP and MAPU Directives

The MAP directives cause a link map of defined symbols that were not purged and of undefined symbols to be written to the list-file (see -COUT in the LINKER command). The MAPU directive is the same, but only applies to undefined symbols.

MAP/MAPU

If MAP is specified, each defined and undefined symbol generated by the linking of object units is listed in the map and preceded by the name of the object unit in which it is located. A map also includes the names of object units that were linked because of embedded Linker directives, and the symbols contained in those object units. If the MAP directive precedes a QUIT directive, the link map will contain all the defined symbols and undefined symbols of the completed bound unit that have not been removed (i.e., purged).

If MAPU is specified, the map contains each undefined symbol and the object unit in which it is located.

MAP and MAPU directives can be interspersed among other Linker directives. When these directives are encountered, all object units named in the link request list are linked before a map is produced. Maps are useful for determining whether all required object units have been linked, and whether all symbols referenced in those object units have been defined.

FORMAT:

$$\left\{\begin{matrix}\text{MAP}\\\text{MP}\\\text{MAPU}\\\text{MU}\end{matrix}\right\}$$

Default: No map produced.

A full link map (a map generated by the MAP directive) comprises the following sections:

START	Address at which execution of the root or overlay will begin; specified in the START directive or in a linked object unit.
LOW	Lowest memory address at which the current root or overlay was based.
HIGH	Next location after the highest address of the current root or overlay.
$COMM	Address assigned to unlabeled COMMON for the bound unit.
CURRENT	Next location after the current address of the root or overlay (when the map was created).
EXT DEFS	All external symbols currently defined in the symbol table.[9]
UNDEF	If an object unit contains no references to undefined symbols, the object unit name is listed and no symbol names are specified. If object units contain references to undefined symbols, the map indicates, for the root and each overlay, the first object unit[10] in which each symbol was referenced and the relative address of the last reference to each symbol; i.e., if an undefined symbol is referenced in the root and an overlay or in two or more overlays, the symbol will appear more than once in the map. The last reference need not be in the same object unit.

[9] Unprotected symbols defined in the root or a previously linked overlay will appear in the map unless the symbols are purged via a PURGE or BASE directive. Symbols erroneously defined as both a value and a location will appear twice under EXT DEFS.

[10] The first reference may occur in the root or a previously linked overlay.

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] symbol name[b]       address[c] or value
[C]       .                    .
          .                    .
          .                    .
[P]*   object unit name     base address of object unit
[P][M] symbol name[b]       address[c] or value
[C]       .                    .
          .                    .                    } OMITTED IF MAPU SPECIFIED
          .                    .
 * *   overlay name         base address of overlay
[P]*   object unit name     base address of object unit
[P][M] symbol name[b]       address[c] or value
[C]       .                    .
          .                    .
          .                    .
[P]*   object unit name     base address of object unit
[P][M] symbol name[b]       address[c] or value
[C]       .                    .
          .                    .
          .                    .
[* * COMMON                 common definitions are separated on the map as well as in the bound]
                            unit when -R is specified
 * * UNDEF
```

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.

[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 object 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

```
LINKER-nnnn-mm/dd/hhmm                          GCOS6 M00400-5100-11/11/0909
MOD= TEST          LINKED ON: 1977/11/23 1518:05.2  -SLIC -M

**  TEST           LINK MAP  1977/11/23 1518:05.2
**START
**LOW      0000  0000
**HIGH     0000  0000
**CURRENT  0000  0000

**EXT DEFS
P    ZMCOMM   0000  0000
P    ZMREL    0000  0000
     XX       0000  0087     VAL      0000

**COMMON
     X        0000  0000


**UNDEF


**  TEST           LINK MAP  1977/11/23 1518:05.2
**START
**LOW      0000  0000
**HIGH     0000  0000
**CURRENT  0000  0000

**EXT DEFS
P    ZMCOMM   0000  0000
P    ZMREL    0000  0000
     XX       0000  0087     VAL      0000
P    CC       0000  0080

**COMMON
P    X        0000  0000


**UNDEF

 VG
CMD ERR

 BBB
CMD ERR


**  TEST           LINK MAP  1977/11/23 1518:05.2



**START
**LOW      0000  0000
**HIGH     0000  0087
**CURRENT  0000  0087

**EXT DEFS
P    ZMCOMM   0000  0000
P    ZMREL    0000  0000
     VAL      0000
P    CC       0000  0080

**COMMON
P    X        0000  0000


**UNDEF


ZLKBGN  77111/
HRS ASSEMBLER 2.50  11/17/77  0816.6 EST THU
```

nnnn is the release identification
mm/dd/hhmm identify the link date and time
This is a SLIC bound unit with separated code.

Map #1
XX is an external location definition
VAL is an external value definition
X is a common location definition

Map #2
The common definition, X, is protected
The external location definition, CC, is defined as XX + hex '6'

Command errors

Map #3
Based at XX
All unprotected location definitions with addresses ≥87 are purged.

LINK ZLKBGN

Figure 2-3. Sample Link Maps

```
**  TEST          LINK MAP  19/7/11/23 1318:05.2
**START   0000  0087
**LOW     0000  0087
**HIGH    0000  0616
**CURRENT 0000  0616

**UNDEF
 *  ZLKBGN  0000  0087
    ZLBEG   0000  0087    EXIT    0000  0089    DSKC10  0000  0547
    FORMCH  0000  017C    TARGET  0000  017A    SAFSW   0000  0187
    MEMNMZ  0000  0180    FORCNT  0000  00F5    MV      0000  0281
    CD      0000  0368    FINEX   0000  0383    ZSW     0000  0193
    ZSW2A   0000  0196    MODE    0000  0318    OLDENT  0000  01E4
    CTOP    0000  01E6    T       0000  01E8    MDCH    0000  01F3
    SDM6    0000  01F7    ENTSZ   0000  01F9    ZMCMAD  0000  01FD
    MDCHIP  0000  01FF    MVWDS   0000  0221    SDAD    0000  020C
    RTMEM   0000  020E    MPDATE  0000  021F    ZDATZ   0000  0226
    CONSOL  0000  02C7    CHKIO   0000  0364    MAP03   0000  0254
    LSTERR  0000  0257    LURNM   0000  0293    OUTNAM  0000  0299
    PORNM   0000  02A5    PRNTCH  0000  02D1    LOGOUT  0000  02D6
    READ    0000  0351    ZLVER   0000  0390    ZLREV   0000  0391
```

Map #4 MAPU

```
**  TEST          LINK MAP  19/7/11/23 1318:05.2
**START   0000  0087
**LOW     0000  0087
**HIGH    0000  0616
**CURRENT 0000  0616

**EXT DEFS
P   ZMCOMM  0000  0000
P   ZMREL   0000  0000
    VAL     0000
P   CC      0000  0080

**  ROOT    0000  0087
 *  ZLKBGN  0000  0087



    ATTACH  0000  03C0    B1      0000  0080    B1M     0000  0088
    B1L     0000  0089    B1R     0000  0087    B2      0000  0113
    B2M     0000  0111    B2L     0000  010F    B2R     0000  010D
    B3      0000  0199    B3E     0000  0219    B3M     0000  0197
    B3L     0000  0195    B3M     0000  0193    BEGIN   0000  0087
    CTLBUF  0000  0223    DATM    0000  03BE    FI6LNG  0000  05C9
    FI6WMR  0000  05C6    FI8LNG  0000  05E8    FI8WMR  0000  05E8
    FIB01   0000  0500    FIB02   0000  0586    FIB03   0000  05A2
    FIB05   0000  0518    FIB06   0000  05C4    FIB08   0000  05E6
    FIB09   0000  0536    FIB10   0000  0551    FIB11   0000  056C
    INP     0000  03CE    INP5    0000  03F1    INP9    0000  0414
    INPA    0000  0437    INPB    0000  045A    INPTM1  0000  03CF
    INPTM5  0000  03F2    INPTM9  0000  0415    INPTMA  0000  0438
    INPTMB  0000  045B    PIFLAG  0000  0386    PTM1    0000  04F6
    PTM2    0000  057D    PTM5    0000  0598    PTM5    0000  0511
    PTM6    0000  0505    PTM8    0000  05F7    PTM9    0000  052C
    PTMA    0000  0547    PTMB    0000  0562    ZLDATE  0000  0392
    VAL2    0002

**COMMON
P   X       0000  0000
 *  ZLKBGN  0000  0087


**UNDEF
 *  ZLKBGN  0000  0087
    ZLBEG   0000  0087    EXIT    0000  0089    DSKC10  0000  0547
    FORMCH  0000  017C    TARGET  0000  017A    SAFSW   0000  0187
    MEMNMZ  0000  0180    FORCNT  0000  00F5    MV      0000  0281
    CD      0000  0368    FINEX   0000  0383    ZSW     0000  0193
    ZSW2A   0000  0196    MODE    0000  0318    OLDENT  0000  01E4
    CTOP    0000  01E6    T       0000  01E8    MDCH    0000  01F3
    SDM6    0000  01F7    ENTSZ   0000  01F9    ZMCMAD  0000  01FD
    MDCHIP  0000  01FF    MVWDS   0000  0221    SDAD    0000  020C
    RTMEM   0000  020E    MPDATE  0000  021F    ZDATZ   0000  0226
    CONSOL  0000  02C7    CHKIO   0000  0364    MAP03   0000  0254
    LSTERR  0000  0257    LURNM   0000  0293    OUTNAM  0000  0299
    PORNM   0000  02A5    PRNTCH  0000  02D1    LOGOUT  0000  02D6
    READ    0000  0351    ZLVER   0000  0390    ZLREV   0000  0391
```

Map #5
External value VAL2 is defined

Figure 2-3 (cont.) Sample Link Maps

```
**  TEST          LINK MAP  1977/11/23 1318:05.2
**START   0000  0087
**LOW     0000  0087
**HIGH    0000  0616
**CURRENT 0000  0616

**EXT DEFS
P   ZHCOMM  0000  0000
P   ZHREL   0000  0000
    VAL     0000
P   CC      0000  008D

**  ROOT    0000  0087
 *  ZLKBGN  0000  0087
    ATTACH  0000  03CD    B1      0000  008D    B1H     0000  0088
    B1L     0000  0089    B1R     0000  0087    B2      0000  0113
    B2H     0000  0111    B2L     0000  010F    B2R     0000  010D
    B3      0000  0199    B3E     0000  0219    B3H     0000  0197

    B3L     0000  0195    B3R     0000  0193    BEGIN   0000  0087
    CTLBUF  0000  0223    DATM    0000  038E    F16LNG  0000  05C9
    F16WHR  0000  05C6    F18LNG  0000  05EB    F18WHR  0000  05E8
    F1B01   0000  0500    F1B02   0000  0586    F1B03   0000  05A2
    F1B05   0000  0518    F1B06   0000  05C4    F1B08   0000  05E6
    F1B09   0000  0536    F1B10   0000  0551    F1B11   0000  056C
    INP     0000  03CE    INP5    0000  03F1    INP9    0000  0414
    INPA    0000  0437    INPB    0000  045A    INPTM1  0000  03CF
    INPTM5  0000  03F2    INPTM9  0000  0415    INPTMA  0000  0438
    INPTMB  0000  045B    PIFLAG  0000  0386    PTM1    0000  04F6
    PTM2    0000  057D    PTM3    0000  0598    PTM5    0000  0511
    PTM6    0000  0505    PTM8    0000  05F7    PTM9    0000  052C
    PTMA    0000  0547    PTMB    0000  0562    ZLDATE  0000  0392

**COMMON
P   X       0000  0000
 *  ZLKBGN  0000  0087


**UNDEF
 *  ZLKBGN  0000  0087
    ZLBEG   0000  0087    EXIT    0000  0089    OSKC10  0000  0347
    FORMCH  0000  017C    TARGET  0000  017A    SAFSW   0000  0187
    MEMNM2  0000  0180    FORCNT  0000  00F5    MV      0000  02B1
    CD      0000  0368    FINEX   0000  0383    ZSW     0000  0193
    ZSW2A   0000  0196    MODE    0000  0318    OLDENT  0000  01E4
    CTOP    0000  01E6    I       0000  01E8    HUCH    0000  01F3
    SDM6    0000  01F7    ENTSZ   0000  01F9    ZHCMAD  0000  01FD
    HUCHIP  0000  01FF    MVWDS   0000  0221    SDAD    0000  020C
    RTMEM   0000  020E    MPDATE  0000  021F    ZDAT2   0000  0226
    CONSOL  0000  02C7    CHKIO   0000  0364    MAP03   0000  0254
    LSTERR  0000  0257    LUNNM   0000  0293    OUTNAM  0000  0299
    PORNM   0000  02A5    PRNTCH  0000  02D1    LOGOUT  0000  02D6
    READ    0000  0351    ZLVER   0000  0390    ZLREV   0000  0391

**********
ROOT  TEST
**********
HIGHEST OVLY    /NUM OF SYMS    0
**********
 SLIC
**********
CMMN  TEST              BASE 0000  0000 ST 0000  0000 -.... HIGH=0000  0077
**********
ROOT  TEST              BASE 0000  0087 ST 0000  0087 -..UI HIGH=0000  0616
**********
*SIZE OF ROOT AND STATIC OVLYS= 0000  0616 HI REL RCD=  14
**********
LINK DONE
**********
```

Map #6
External value VAL 2 is purged by a VPURGE directive

QUIT directive entered

This information always appears on a list file.

Figure 2-3 (cont.) Sample Link Maps

ARGUMENT DESCRIPTION:

name
Name of the nonfloatable 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	
OVLY A2	Designates the end of the root (which comprises object units A.O and B.O) and specifies that the next overlay is a nonfloatable overlay named A2. The Linker assigns the number 00 to this overlay.
LINK X	
LINK Y	
MAP	
QUIT	

PROTECT Directive

The protect directive prevents certain symbols and/or object unit names from being removed from the symbol table. Symbols that identify addresses from the first operand through the second operand are protected, and object unit names equated to addresses within that range are protected. If a second operand is not specified, the symbol at the address of the first operand and any other symbols or object unit names equated to that address are protected. Once a symbol or object unit name is protected, it cannot later be purged.

FORMAT:

$$\left\{\begin{matrix}\text{PROT}\\\text{PT}\end{matrix}\right\}\ \left\{\begin{matrix}\$\\\%\\\text{X'address'}\\\text{=object-unit-name}\\\text{xdef}\\\#\end{matrix}\right\}\left[,\left\{\begin{matrix}\$\\\%\\\text{X'address'}\\\text{=object-unit-name}\\\text{xdef}\\\#\end{matrix}\right\}\right]$$

ARGUMENT DESCRIPTIONS:

$
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
Address of any previously defined external symbol.

#
The current address.

Example 1:

```
PROT X'1234',X'4565'
```

This directive protects symbols and object unit names that identify addresses from 1234 through 4565.

Example 2:

```
PT =FIRST
```

This directive protects symbols that identify the base address of the object unit FIRST, and all symbols equated to that address. The base address of FIRST is determined by producing a link map (see "MAP and MAPU Directives").

Example 3:

```
PROT SYM,X'5555'
```

This directive protects symbols that identify addresses from the address of the previously defined external symbol named SYM through 5555; object unit names equated to those addresses also are protected.

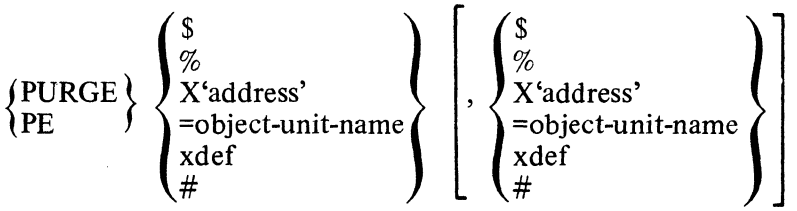
PURGE Directive

The PURGE directive causes the Linker to remove from the symbol table unprotected symbols that define addresses from the first operand through the second operand, and/or object unit names equated to addresses within that range. If a second operand is not specified, the symbol at the address of the first operand and any other symbols or object unit names equated to that address are purged.

An object unit currently being linked may contain definitions used for previously linked object units that won't be used for subsequent object units to be linked. By removing from the symbol table symbols that are no longer required, there is more room for symbols that will be required by subsequently-linked object units.

NOTES:
1. Undefined symbols cannot be purged.
2. Symbols and object unit names that are protected by a protect directive cannot be purged.
3. Only symbol addresses (not values) can be purged.

FORMAT:



ARGUMENT DESCRIPTIONS:

$
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
Address of any previously defined external symbol.

#
The current address.

Example 1:

```
PURGE X'1234',X'4565'
```

This directive purges unprotected symbols that identify addresses from 1234 through 4565, and unprotected object unit names equated to addresses within that range.

Example 2:

```
PE =FIRST
```

This directive purges unprotected symbols that identify the base address of the load unit FIRST, and any other unprotected symbol names equated to that address. The base address of FIRST is determined by producing a link map (see "MAP and MAPU Directives").

Example 3:

```
PURGE SYM,X'5555'
```

This directive purges unprotected symbols that identify addresses from the address of the previously defined external symbol SYM through 5555; unprotected object unit names equated to addresses within that range also are purged.

QUIT Directive

The QUIT directive designates that the last Linker directive has been entered. Specify QUIT after the last overlay, or at the end of the root if there are no overlays.

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.

The QUIT directive is required.

FORMAT:

```
{QUIT}
{QT  }
{Q   }
```

SHARE Directive

The SHARE directive designates that the bound unit is shareable; i.e., it will be loaded into the system pool. If another task requests that the bound unit be loaded, instead of another copy of the bound unit being loaded, the existing copy in memory is used. The bound unit *must* have reentrant code, but the system does not check to see that it does.

SHARE must be specified in the definition of the root before the first overlay is defined.

SHARE directives can be embedded in assembly language CTRL statements.

FORMAT:

```
{SHARE}
{SE   }
```

START Directive

The START directive designates the relative location within a root or overlay at which execution of the root or overlay will begin once it is loaded into memory by the Loader.

If a linked object unit contains a start address (an Assembler or compiler END statement was specified) and the START directive is specified, the first start address encountered (in either a START directive or an END statement) is used by the Linker for that root or overlay.

FORMAT:

```
{START} symbol
{ST   }
```

ARGUMENT DESCRIPTION:

symbol

Name of the external symbol whose address designates the relative address at which the root or overlay will begin executing.

Default: Start address specified in the first linked object unit that has a start address. If the symbol is never defined or a start address is not found, the start address is relocatable 0.

SYS Directive

The SYS directive designates that the bound unit being created can be used as a system task in the system task group. To use the bound unit in a system task group, it must be loaded during system configuration using the LDBU configuration directive, which is described in the "Startup and Configuration Procedures" section of the *System Building* manual. If SYS is not specified, the CLM Loader will not load the bound unit. The SYS directive can be embedded in assembly language CTRL statements.

FORMAT:

```
SYS
```

Example:

```
SYS
```

If source units are written to create a function not provided by the operating system and the SYS directive is specified during linking, the bound unit created can be loaded during system configuration and the capability it provides can be used.

VAL Directive

The VAL directive creates a value definition at link time which is equivalent to the difference between two external location definitions.

FORMAT:

VAL value-definition, external-location$_1$ – external-location$_2$

ARGUMENT DESCRIPTIONS:

value-definition
Assigns a name to the value of the distance between two locations.

external-location$_n$
A location defined externally.

VDEF Directive

The VDEF directive assigns a *value* to an external symbol. A symbol should be defined only once, as a value or as a location. When a symbol is defined, its definition is put into the Linker symbol table so that it can be used during linking to resolve external references.

FORMAT:

$$\left\{\begin{matrix}\text{VDEF}\\ \text{VF}\end{matrix}\right\} \text{ symbol,X'value'}$$

ARGUMENT DESCRIPTION:

symbol
One to six alphanumeric characters.

value
Value of the designated symbol; must be a one-word hexadecimal integer enclosed in single apostrophes and preceded by X.

Example:

```
VDEF  XMP,X'12'
```

This directive assigns the value 12 to the symbol XMP.

VPURGE Directive

The VPURGE directive causes the removal of the specified external value definition.

FORMAT:

VPURGE value-definition

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.
BASE =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.
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.)

Example:

Described below are three methods of relocating a unit so that it is executed at relative location 100 within the memory assigned to the bound unit. This unit will constitute a root.

	Assembly	*Linking*
Method I:	ORG X'0100' before the first line of executable code.	Don't specify BASE directive.
Method II:	Don't specify ORG. (Default is 0.)	Specify BASE X'100'
Method III:	ORG X'10'	BASE X'F0'

6. When relocating object units or nonfloatable overlays during the assembly or linking procedure, it is your responsibility to ensure that code is not inadvertently overwritten.
7. If more than six characters are specified for an object unit name or symbol name, or more than 12 characters are designated for a bound unit name, subsequent characters are truncated.
8. Forward external references with offsets are not permitted. If the Linker encounters them, an error message is issued and execution of the Linker terminates.
9. Common definitions may appear more than once in object units being linked. Only the first occurrence of either a labeled or unlabeled common definition block is used to reserve the defined amount of memory. Therefore, the largest definition of labeled or unlabeled common should be linked first. Common blocks are allocated space by the Linker by assigning the current location counter (address) to the symbol name, and then incrementing the current location counter by the size of memory specified for the common block.
10. A BASE directive in the root or an overlay cannot specify an address less than the beginning of that root or overlay; i.e., it cannot be less than the first word of the first object unit linked in that root or overlay.
11. If BASE $ or BASE % is specified in the root, it is equivalent to BASE 0.
12. The start address of the root or an overlay must be in the first 32K-1 words.
13. Preallocated bound unit files and preallocated work files will decrease the execution time of the Link session. If the -W command line argument is specified for the first link of a bound unit, the work files will be saved and reused for subsequent links.

SECTION 3
PROGRAM EXECUTION

This section presents a summary of the procedures followed and the commands used in executing a program.

DESIGNATING FILES

Prior to executing an application program, an ASSOC (associate) or GET command must be entered to associate the program's logical file numbers (LFNs) with physical files. For a detailed description of the ASSOC and GET commands, see the *Commands* manual.

ASSOC Command

The ASSOC command establishes the relationship between a specified pathname of a file in a task and the LFN by which the task refers to this file.

FORMAT:

ASSOC lfn path

ARGUMENT DESCRIPTION:

lfn
The logical file number by which a task is to refer to a file.

path
The pathname of the file to which the task is to refer.

GET Command

The GET command reserves a file system entity (i.e., a tape or disk file or volume, a disk directory or file, or a card, print, or terminal device file) and establishes an association between the reserved entity and a logical file number, if such an association does not already exist.

FORMAT:

GET path lfn [ctl_arg]

ARGUMENT DESCRIPTION:

path
The pathname of the file being reserved; can be any valid pathname form for file- or volume-level access.

lfn
The logical file number (lfn) by which this file is referenced during access.

[ctl_arg]
Control arguments used in the reservation of a file are described under the GET command in the *Commands* manual.

SETTING SWITCHES

The order of execution of a task group can be controlled by optionally using the MSW command (e.g., for an RPG program).

MSW Command

The MSW command modifies selected external switches associated with the issuing task group to control the execution of that task group.

FORMAT:

MSW ctl_arg

ARGUMENT DESCRIPTION:

ctl_arg
One or more control arguments chosen from the following list:

-ON $S_i[S_i]$ ...
Set the external switch indicated by S_i ON. Each S_i is a hexadecimal digit from 0 through F.

-OFF Si[Si] ...
Set the external switch indicated by S_i OFF. Each S_i is a hexadecimal digit from 0 through F.

-ALL v
Set all switches to the value v. The value v can be either ON or OFF.

REQUESTING PROGRAM EXECUTION

Programs can be prepared and executed in the same or in different task groups. The following paragraphs summarize the appropriate methods of initiating program execution in each case.

Program Preparation and Execution In the Same Task Group

Assume that the program preparation is done in a task group with the command processor as the lead task. If the task group memory pool is large enough to accommodate the application program, program execution can be initiated simply by entering the bound unit pathname. For example, if the bound unit's relative pathname in the working directory is INVENT, enter INVENT.

Program Execution In a Different Task Group From Program Preparation

If the application program cannot run in the task group used for program preparation, e.g., if the memory pool is too small or if the lead task is not the command processor, then a new task group must be created.

This is accomplished from an existing task group that has the command processor as its lead task by using one of the following procedures:

1. The Create Group (CG) command followed by the Enter Group Request (EGR) command
2. The Spawn Group (SG) command

If the installation supports the LOGIN function, then the application program can be executed using the LOGIN command.

For detailed descriptions of these commands, see the *Commands* manual.

Using the CG and EGR Commands

The CG and EGR commands must be used in conjunction with each other. A description follows on the purpose and use of each command.

CG Command

The CG command allocates and creates the data structures required to define and control the execution of an online task group. The -EFN argument specifies the name of the bound unit to be executed.

FORMAT:

CG id base-lvl [ctl_arg]

ARGUMENT DESCRIPTION:

id
The group identification of the new task group. It is a two-character name that cannot have the $ as its first character.

base_lvl
A base priority level, relative to the highest system level, at which all tasks in this task group will execute.

[ctl_arg]
One or more control arguments chosen from the following list.

{-EFN root
-EFN root?entry}
The name of a bound unit root segment to be loaded as the lead task if it is not already loaded. The root segment name can be suffixed with ?entry, where entry is a symbolic start address within the root segment. If ?entry is not given, the start address established when the bound unit was linked is assumed.

-ECL
The root segment of the command processor is to be loaded as the lead task.

-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
Specifies the highest logical file number used by any task in the task group. The maximum value is 255. If -LFN is not specified, n assumes the value 15.

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

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

EGR Command

The EGR command activates the lead task of an online task group previously created by a CREATE GROUP command.

FORMAT:

EGR id [in_path] [ctl_arg]

ARGUMENT DESCRIPTION:

id
The group identification of a task group previously created by a CG command specifying the same id.

[in_path]
The name of the file from which commands and user input are to be read by the task group during execution. This argument is set to null if it is not specified. It is required if the CG command specified the control argument -ECL.

[ctl_arg]
One or more control arguments chosen from the following list.

-OUT out_path
Defines the path name of the file which is to receive user output and error output from the task group. If not specified, one of the following assumptions is made:

If in_path specifies a disk file, out_path = in_path.AO
If in_path specifies an interactive terminal, out_path = in_path
If in_path is not specified, out_path is null
If in_path specifies an input-only device, out_path is null.

-WD path
Specifies that path is to be used as the working directory pathname.

-ARG arg arg . . . arg
Indicates that additional arguments required by the task group during execution follow. These additional arguments are passed to the lead task 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.

Using the SG Command

The spawn group command creates, requests the execution of, and then deletes a task group.

FORMAT:

SG id base_lvl [in_path] [ctl_arg]

ARGUMENT DESCRIPTION:

id
The group identification of the task group to be spawned. It is a two-character name that cannot have the $ as its first character.

base_lvl
A base priority level, relative to the highest system level, at which all tasks in this task group will execute.

in_path
The name of the file from which commands and user input are to be read by the task group during its execution. The file name is set to null if the in_path argument is not specified; in_path must be specified if the control argument -ECL (see below) is used or implied.

[ctl_arg]
One or more control arguments chosen from the following list:

-OUT out_path
Defines the pathname of the file which is to receive user output from the task group. If not specified, one of the following assumptions is made:

If in_path specifies a disk file, out_path = in_path.AO
If in_path specifies an interactive terminal, out_path = in_path
If in_path is not specified, out_path is null
If in_path specifies an input-only device, out_path is null.

-WD path
Specifies that path is to be used as the working directory pathname.

{-EFN root | -EFN root?entry}
The name of a bound unit root entry which is to be loaded as the lead task. The root segment name can be suffixed with ?entry, where entry is a symbolic start address within the root segment. If ?entry is not given, the start address established when the bound unit was linked is assumed.

-ECL
The root segment of the command processor is to be loaded as the lead task.

-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
Specifies the highest logical file number used by any task in the spawned task group. (The maximum value is 255). If -LFN is not specified, n assumes the value 15.

-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. When the login procedure has been executed, it spawns the task group to be associated with the user's terminal. Once having access to the system, the user can only re-invoke login after first issuing the BYE command. The user employs the LOGIN command from a direct-login terminal.

FORMAT:

L [login_id] [destination_id] [ctl_arg]

ARGUMENT DESCRIPTION:

login_id

Identifies the user who is attempting to access the system. Provides an identification for the spawned task group. The login_id argument has one of the following forms:

person	One to 12 alphanumeric characters, beginning with a letter.
person.account	One to 12 alphanumeric characters, beginning with a letter.
person.account.mode	Mode is one to five alphanumeric characters, beginning with a letter.

The login_id argument is not specified if the terminal allows the login procedure to be initiated by abbreviation. In this case, a single character is typed following the L.

destination_id

Optionally allows the user to specify additional information about the login procedure. This information is not processed by the system. The destination_id has one of the following forms:

field_c	One to five alphanumeric characters.
field_b.field_c	Field b is decimal; 0 through 65536.
field_a.field_b.field_c	Field a is decimal; 0 through 65536.

ctl_arg

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

-PO path [endpoint]

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.

endpoint

Group id/pool id of the spawned task. The group id and the pool id are represented by the same two-character value. If this argument is not specified, the group id is a two-character value whose left (first) character was generated by the login procedure while connecting the terminal 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 4
PATCH

The Path utility program applies patches to and removes patches from object units and bound units. Patches are identified by patch-id's; (Patch also can be used to list, by patch-id,) all patches for an object unit or bound unit. The listing is written to the user output file.

Execution of Patch is controlled by directives entered to Patch through the operator's terminal, another terminal, or a card reader. Each directive is listed below, followed by its function; these directives are described in detail later in this section.

Directive Name	*Function*
EP	Eliminate patch(es)
HP	Apply hexadecimal patch(es)
DP	Data patch
LP	List patches
Q	Execute previously specified Patch directives, and then terminate execution of Patch
*Δ	List a comment on the user output file.

LOADING PATCH

To load Patch, enter the PATCH command, as follows:

FORMAT:

PATCH filenm[ctl_arg]

ARGUMENT DESCRIPTIONS:

filenm
Pathname of the object unit file or bound unit file to be patched. If an object unit is being patched, the last two characters of the pathname must be .O.

ctl_arg
The following control arguments may be entered:

-IN path
Pathname of the device through which Patch directives will be entered; can be the operator's terminal, another terminal, or a card reader. Error messages are written to the error output file. Patch 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.

PROMPT argument

{-PROMPT}
{-PT}

If input is from the operator terminal or another terminal, each time the PATCH utility program is ready to accept an input line the typeout P? appears on the input device.

Default: The string P? is not printed.

SUBMITTING PATCH DIRECTIVES

Each Patch directive consists of only a directive name or a directive name followed by one or more parameters. If one or more parameters are to be specified in a Patch directive, the directive name must be immediately followed by a space. Each parameter, except for the last, must be followed by a comma.

Multiple Patch directives may be specified for the file named in the filenm agrument of the PATCH command.

Patch directives may be entered in any order, except for quit, which must be entered last.

If directives are being entered through the operator terminal or another terminal, press RETURN at the end of each line. Each time RETURN is pressed, except after quit, the typeout P? is reissued if the Prompt Control argument was specified in the command line.

To enter Patch directives for a different file, you must reload Patch, specifying a different file in the filenm argument.

PATCHING TECHNIQUES

Naming the Patch

Each patch has a patch-id by which it is identified. When you designate in hexadecimal patch (HP) directives that one or more patches are to be applied to a specified object unit or bound unit, you must specify a patch-id; the patch-id identifies the patch(es) and designates whether the patch(es) are to be applied to an object unit or to the root, or to an overlay of a bound unit. To eliminate patches from an object unit or bound unit, you must specify in the eliminate patch directive the patch-id with which the patch(es) are associated. See "Hexadecimal Patch Directive (HP)" for a description on how to designate patch-id's.

Applying the Patch

If an object unit is being patched, object records are created for the specified patches and appended to the end of the object file. When the object unit is processed by the Linker, existing values are replaced with the specified patch values.

If a bound unit is being patched, each specified patch value is applied directly to the proper image record in the bound unit. The previous value, the patch-id, and the patch value are saved in a Patch history record that is written at the end of the file area allocated to the bound unit. This record is referred to each time a list patch directive or eliminate patch directive is specified.

NOTE: Use caution when patching running programs. If a program or one of its overlays is loaded while in the process of being patched, results are unspecified.

PATCH DIRECTIVES

Data Patch Directive

For bound units the data patch directive (DP) applies one or more hexadecimal patches, by relative location, to the data section of the bound unit. The bound unit must have separate code and data sections.

For object files, the DP directive causes patches to be applied to common areas.

FORMAT:
(For bound units)

DP patchid Δ/adr$_1$ Δ patchval$_{11}$ [Δ patchval$_{12}$ . . . Δ patchval$_{1n}$]

[Δ/adr$_2$ Δ patchval$_{21}$ [Δ patchval$_{22}$. . . Δ patchval$_{2n}$]]

(For object files – local common block)

DP patchid Δ/offsetl$_1$ Δ patchval$_1$ Δ/offsetl$_2$ Δ patchval$_2$

(For object files – named common block – one blockname per directive)

DP patchid Δ blockname Δ/offset Δ patchval$_{11}$ [Δ patchval$_{12}$. . . patchval$_{1n}$]

ARGUMENT DESCRIPTIONS:

patchid
: Eight-character patch-id, which identifies the subsequent patch(es). The last two characters must identify the root or overlay to which the patch(es) are being applied. If an object unit or the root of a bound unit is being patched, the last two characters must be RT. If an overlay is being patched, the last two characters identify the hexadecimal overlay number; the first overlay is 00, and subsequent overlays are numbered consecutively in ascending order. There may be no embedded blanks. Within the root and each overlay, patch-id's must be unique.

/adr$_n$
: Relative location at which the first (or only) subsequent patch value will be applied. Each address must comprise one to six right-justified, hexadecimal characters, and must be preceded by the slash character /. Subsequent patch values, if any, are applied to succeeding memory locations.

NOTE: Care must be taken in specifying an address to be patched in either an object unit or a bound unit. If the address of a location to be patched is identified when a bound unit is being executed, that memory address contains three possible factors:

1. The original address of the location in the bound unit relative to the beginning of the bound unit
2. The linking relocation factor
3. The loader relocation factor

If the address is identified at execution time and the bound unit is to be patched, the loader relocation factor must be subtracted from the address identified in the executing bound unit. If the object unit is to be patched, both the linking and loader relocation factors must be subtracted. Object unit locations can also be obtained through examination of the listing produced during assembly of the object member.

offsetl_1 ,offsetl_2
Non-negative offset from the beginning of $LCOMW.

patchval$_n$
Specifies a value of one to six hexadecimal characters to insert into $LCOMW. Relocatable values are *not* permitted and only one patch value can be specified for each patch.

blockname
Symbolic name of the common block. The name can contain one to six characters.

offset
Offset from the symbol name of the common block.

/patchval$_{1n}$
The value to be inserted at an address, replacing the contents of that location. The value must be specified as one of the following:

1. Data, represented by one to four hexadecimal characters.
2. Relocatable address, represented by one to six hexadecimal characters, preceded by the character <.

Eliminate Patch Directive

The eliminate patch directive (EP) eliminates all patches associated with a specified patch-id from the designated object unit or bound unit. The patch(es) must have been previously applied by a hexadecimal patch (HP) directive. To determine what patches have been applied, and their patch-id's, enter the list patch (LP) directive, which is described later is this section.

FORMAT:

EP patchid

ARGUMENT DESCRIPTION:

patchid

Patch-id of the patch(es) to be removed. A patch-id comprises eight characters; the last two characters must identify the root or overlay to which the patch(es) are being applied. If an object unit or the root of a bound unit is being patched, the last two characters must be RT. If an overlay is being patched, the last two characters identify the hexadecimal overlay number; the first overlay is 00, and subsequent overlays are numbered consecutively in ascending order. There may be no embedded blanks. Within the root and each overlay, patch-id's must be unique.

Hexadecimal Patch Directive

The hexadecimal patch directive (HP) applies one or more individual patches, by relative location, to an object unit or bound unit.

If a *bound unit* is being patched, you can designate that specified patch(es) be applied only if specified location(s) currently contain specified value(s); these are called verification values. Within a single HP directive, verification values may be specified for some or all of the locations. If *any* of the verification values do not match the values currently at the locations for which verification values were specified, *none of* the patches specified in the HP directive are applied.

FORMAT:
Without Verification Values:

HP patchid,/adr_1,$patchval_1$ [,$patchval_2$ . . . $patchval_n$] [,/adr_2,$patchval_1$,$patchval_2$ . . .

$patchval_n$] . . .

With Verification Values:

HP patchid,/adr_1,($verval_1$ $patchval_1$ [,$verval_n$,$patchval_n$])[,/adr_2,($verval_1$,$patchval_1$

[,$verval_n$,$patchval_n$)]

NOTES:

1. One or more lines of parameters may be specified. When two or more lines of parameters are entered for an HP directive, the last character on each line must be a valid hexadecimal character. Individual fields, values, and addresses must not be split between lines. The entry of a patch directive name (e.g., EP, LP) at the beginning of a line designates the end of the previous patch directive.
2. A space may be used in lieu of a comma as a separator.

ARGUMENT DESCRIPTIONS:

patchid
: Eight-character patch-id, which identifies the subsequent patch(es). The last two characters must identify the root or overlay to which the patch(es) are being applied. If an object unit or the root of a bound unit is being patched, the last two characters must be RT. If an overlay is being patched, the last two characters identify the hexadecimal overlay number; the first overlay is 00, and subsequent overlays are numbered consecutively in ascending order. There may be no embedded blanks. Within the root and each overlay, patch-id's must be unique.

/adr_n
: Relative location at which the first (or only) subsequent patch value will be applied. Each address must comprise one to six right-justified, hexadecimal characters, and must be preceded by the character /. Subsequent patch values, if any, are applied to succeeding memory locations.

NOTE: Care must be taken in specifying an address to be patched in either an object unit or a bound unit. If the address of a location to be patched is identified when a bound unit is being executed, that memory address contains three possible factors:

1. The original address of the location in the bound unit relative to the beginning of the bound unit.
2. The linking relocation factor
3. The loader relocation factor

If the address is identified at execution time and the bound unit is to be patched, the loader relocation factor must be subtracted from the address identified in the executing bound unit. If the object unit is to be patched, both the linking and loader relocation factors must be subtracted. Object unit locations can also be obtained through examination of the listing produced during assembly of the object member.

$patchval_n$

The value to be inserted at an address, replacing the contents of that location. The value must be specified as one of the following:

1. Data, represented by one to six hexadecimal characters
2. Relocatable address, represented by one to six hexadecimal characters, preceded by the character <

$verval_n$

Verification value; one to four hexadecimal characters specifying value that currently should be in location at which subsequent patch will be applied.

NOTES:

1. Each verval must be immediately followed by a patchval (see above).
2. The verification value(s) and patch value(s) associated with each address parameter must be enclosed within parentheses.

Example 1:

```
HP PTCHIDRT,/1B2A,1FFF,1DFC,<2BFC,2D4E,<ABF2
```

This hexadecimal patch (HP) directive requests that the subsequent patches, identified by the name PTCHIDRT, be applied to the root. Patches $1FFF_{16}$ through $<ABF2_{16}$ are to be inserted in successive locations, with the first patch $1FFF_{16}$ to be located at address $1B2A_{16}$. The hexadecimal patches are to replace any previous values in these locations. The value to be inserted in address $1B2C_{16}$ is the global address $2BFC_{16}$, which is to be relocated at load time; the relocatable address $ABF2_{16}$ is to be inserted in address $1B2E_{16}$.

Example 2:

```
HP VPATCH01/1FEA,(1A1,1B7,1A7,1B8),/1E72,8900
```

This example illustrates the usage of verification values in a hexadecimal patch (HP) directive requesting that specified patches, identified by the name VPATCH01, be applied to overlay 01. Patch will check location $1FEA_{16}$ for the value $1A1_{16}$, and location $1FEB_{16}$ for the value $1A7_{16}$; if the values are at those locations, then the contents of locations are changed as follows: location $1FEA_{16}$ will contain $1B7_{16}$, location $1FEB_{16}$ will contain $1B8_{16}$, and location $1E72_{16}$ will contain 8900_{16}. If either of the verification values is incorrect, none of the three locations will be changed.

List Patches Directive

The list patches directive (LP) produces a listing of all patches within the object unit or bound unit being patched. The listing is produced on the user output file.

If a bound unit is being patched, the listing designates, for each patch, the following information in the order listed: full patch-id, address at which the patch was applied, contents of the location before the patch was applied, and the patch value.

NOTES:

1. In the listing, the last two characters of each patch-id (i.e., the characters that identify the root or overlay) appear *first,* and are separated from the other characters constituting the id by spaces. When a bound unit is being patched in a common area, the letters CM are printed rather than RT.
2. If termination of the listing of patches is desired before normal completion of the list process, use the BREAK facility followed by a NEW_PROC command. The PATCH program must then be reloaded.

FORMAT:

LP

Example:

01 NOHLT3 02E2 0000 0F02

The above printout is one line of a listing of patches applied to a bound unit being patched. The printout has the following meaning: A patch identified by the patch-id NOHLT301 was applied to overlay 01. The patch was applied to location 02E2; this location previously contained 0000, and now contains 0F02.

If an object unit is being patched, the listing designates, for each patch, the following information in the order listed: patch-id (excluding the last two characters, which identify the root), address at which the patch was applied, and the patch value.

Example:

NUMBRF	0162	0444
	0163	0222
NUMBRH	01A6	0333
	01A7	0444
	01A8	<0221
	01A9	0004
	01AA	<0321

The above typeout is a listing of patches applied to an object unit being patched. The first line designates that patch 0444, whose patch-id is NUMBRF, was applied to location 0162. Note that the last two characters of the patch-id (e.g., RT) were omitted from the printout.

Quit Directive

The quit directive (Q) informs Patch that the last Patch directive has been entered, and initiates processing of the specified Patch directives. This directive should be preceded by at least one other Patch directive. When the directive(s) have been executed, execution of Patch terminates.

FORMAT:

Q

Comment Directive

The comment directive (*Δ) causes the accompanying text to be listed on the user output file. The contents of the comment directive are not saved.

FORMAT:

*Δcomment-text

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 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.

TABLE 5-1. SYMBOLS USED IN DEBUG DIRECTIVE LINES

Symbol Type	Meaning
Arithmetic Operators	
plus sign (+)	Performs addition.
minus sign (–)	Performs subtraction.
K	Multiplies a hexadecimal integer by 1024 decimal (400 in hexadecimal) when K is the last character of an integer expression.
Address Operators	
period (.)	Represents the last start address used in a previous memory reference directive (DH,CH,DP).
ampersand (&)	Represents the address of the next location beyond the last one used by a previous memory reference directive (DH,CH,DP).
brackets [ ] [a]	Signifies the contents of the location defined by the expression within the brackets. Three levels of nesting may be used.
Reserved Symbols	
$Bn	Contents of base register n of the active level. The values 1 through 7 can be used for n.
$Rn	Contents of the data register n of the active level. The values 1 through 7 can be used for n.
$P	Contents of the program counter of the active level.
$I	Contents of the indicator register of the active level.
$IV	Address of the Task Control Block of a trapped task which is currently at the head of the active level's trap queue.
$S	Contents of the system status register (level number and privilege bit only) of the active level.
$SL	Represents the value of the level number of the active level.
$E	Used with set bound units breakpoint directive to represent the entry point as defined in the bound unit or by the caller. Used in place of $P associated with ordinary breakpoints.
G through Z	Twenty single-character symbols having initial values of zero. Values may be assigned using the AS directive.
Notational Symbols	
braces { } [a]	For a single enclosed argument, indicates that the argument is optional. If more than one argument is enclosed by the braces in a vertical listing, the braces indicate a choice is to be made. In this case, optional arguments are identified in the text.
ellipses ...	Indicates the ability to repeat within braces.
delta (Δ)	Indicates one or more spaces.
Vertical bar |	Indicates a choice between two or more arguments.
Debug Language	
{^}< {^}= {^}>	Specify the condition to be satisfied in an IF directive for continual processing of the directive line. {^} indicates a logical 'NOT' which may optionally be used.
parentheses ( )	Indicate directive or header information to be stored for later use. Unmatched right parenthesis results in an error. A right parenthesis that is paired with the first left parenthesis terminates the directive definition.
exp	Indicates a valid expression formed using expression elements.
rexp	Consists of exp_1/exp_2, where exp_1 is a hexadecimal number that is a value or a location; exp_2 is an optional hexadecimal repeat factor whose value must be between 1 and 32,767. If exp_2 is omitted, the value of exp_1 is assumed.
;	Separation character between directives on the same line.
*	Signifies "all" in certain print, clear, and list directives.

[a] In this section, brackets and braces have special meanings, as described above. In each other section, they are interpreted differently (see "Symbols Used in This Manual," in the "Overview" section).

Table 5-2 summarizes Debug directives by function. These directives are described in detail alphabetically on the following pages. In each directive's format, it is assumed that Debug was previously designated as the OIM default task group when the operator terminal is specified as user-in, so Δ$DΔ is not specified before each directive name.

NOTES:

1. Pay careful attention to the format of each directive, because the usage of delimiters, if any, between a directive name and the first (or only) parameter varies according to which directive is being specified.
2. If a directive has a parameter in which you may specify the logical resource number (lrn) of the device on which information will be printed, Debug uses the specified device without first determining whether the device has been reserved for exclusive use by another task; i.e., Debug bypasses the file system.

Planning Considerations

Setting Breakpoints

Breakpoints can be set to trap at selected task code locations. At breakpoints, memory and register values can be displayed and changed. In this way, a task can be executed, the values of its variables checked as execution proceeds, code modified, and if necessary, variable values changed in order to test the sequence of code up to the next breakpoint.

Following are guidelines for setting breakpoints:

1. Breakpoints can be set in a task group (or in an overlay in a task group) only when the task group/overlay currently is memory resident. The SB (set bound unit breakpoint) directive should be used to gain control of a task group bound unit/overlay when it is loaded, to allow ordinary breakpoints to be properly set.
2. Breakpoints may *not* be set in code that will be executed at the inhibit level.
3. If shareable code contains breakpoints, each task that uses the code encounters the breakpoint, regardless of which task group the task is in.

Breakpoints are set in task groups by specifying the set breakpoint directive (Sn); the detailed description of Sn includes additional rules for specifying breakpoints.

Controlling Output Using a Breakpoint

Output can be redirected from an operator terminal by using a breakpoint. When the breakpoint condition occurs, the FO directive can be used to redirect the Debug output.

In the discussions which follow, the terminology "current Debug output device" refers to either an interactive terminal specified for a Task Group or the device defined by an FO directive.

Determining/Setting the Active Level

The active level is the priority level currently in effect. Directives relating to specific task context are effective only on the active level. You must establish a level as the active level by specifying the set level directive before using these directives from the directive input device. Thereafter, the active level assumes the value that will most probably be needed, based on the Debug action in progress; i.e., breakpoint, trace trap, or temporary reference to a different level.

If you want to reference specific task context on another priority level from the directive device, you can change the active level by respecifying the set level directive (SL) or *temporarily* designate another level as the active level by specifying the set temporary level directive (TL); in the latter case, the level is considered the temporarily active level. After the desired actions are performed on the temporarily active level, the active level reverts to the level specified in the previous set level directive.

TABLE 5-2. SUMMARY OF DEBUG DIRECTIVES, BY FUNCTION

Function	Directive Name	Meaning
Directive line definition and handling	Dn	Define directive line n
	En	Execute directive line n
	P*	Print all predefined directive lines
	Pn	Print directive line n
Breakpoint control	C*	Clear all breakpoints
	Cn	Clear breakpoint n
	GO	Proceed form breakpoint
	L*	List all breakpoints
	Ln	List breakpoint n and associated directive line
	Sn	Set breakpoint n
Bound unit breakpoint control	CB*	Clear all bound unit breakpoints
	CBn	Clear breakpoint n in bound unit
	LB*	List all bound unit breakpoints
	LBn	List breakpoint directive lines for bound unit
	SBn	Set bound unit breakpoint n
Trace trap control	DT	Define trace directive line
	PT	Print trace directive line
	ST	Start-j-mode trace
	ET	End j-mode trace
Active level control	SL	Set active level
	TL	Set temporarily active level
Memory and register control	AR	Print contents of all active level registers
	CH	Change memory
	DH	Display memory in hexadecimal
	DP	Dump memory in hexadecimal and ASCII
Symbol control	AS	Assign a hexadecimal value to symbol
	VH	Print value of expression in hexadecimal
General execution	FO	Redirect output
	Hn	Print header line
	IF	Conditional execution
	LL	Specify line length of operator's terminal or another terminal currently in use
	RF	Reset file location
	SF	Specify file location
	QT	Abort Debug task group

NOTES: 1. The memory and register control directives (AR, CH, DH, and DP) apply to registers on the active level. To determine which level is the active level and/or to set the active level to a specified value, see "Determining/Setting the Active Level" below.

2. The following directives are predefined or delayed execution directives and directive lines associated with them are stored in the file DEBUG, WORK: Sn, Hn, Dn, DT, SBn.

Following are guidelines for determining which level is the active level, and methods of setting the active and temporarily active level.

1. The set level directive (SL) sets (or changes) the active level. The specified level becomes the default level accessible by the operator terminal or another terminal that is the directive input device.
2. The set temporary level directive (TL) designates a level as the temporarily active level; this permits you to display or alter registers of a level different from the default terminal level without permanently changing the default terminal level.
3. Whenever a break or trace point is processed for a task, the active level is set to the level of that task for the duration of any stored-directive line execution. After this duration, the last operator-specified value of "active level," if any, is again in force.

Maintaining a Trace History

When using Debug with disk-stored directive lines that execute upon encountering a trap or a breakpoint, a trace history may be maintained on a line printer.

Also, while at a Debug breakpoint from a given breakpoint stall, a particular task may be set to run in jump-trace mode. In this case, every departure from the current sequence of instructions generates a trace trap.

All Registers Directive

The all registers directive (AR) causes the printing of all registers for the active level.

FORMAT:

AR {/lrn}

ARGUMENT DESCRIPTION:

/lrn
Logical resource number of the device on which the printout will occur.
Default: Current Debug output device

Example:

```
AR/3
```

This example causes the contents of all the registers for the *active* level to be printed on the device referred to as logical resource number 3.

NOTE: References to registers on the active level are valid only if the task has come to a breakpoint. The AR directive cannot be used otherwise (e.g., for tasks that are suspended or that have executed a trap that produced a default trap handler error).

Assign Directive

The assign directive (AS) assigns a specified hexadecimal value to a specified symbol; this directive is used to alter registers of the active level, and to define reserved symbols. Bound unit breakpoints lie within the Exec Loader, not in your task context. As a result, the assign directive is refused by Debug, if the current level's task is stalled on a bound unit breakpoint.

FORMAT:

ASΔsymΔexp {ΔsymΔexp...}

ARGUMENT DESCRIPTIONS:

sym
Register or reserved symbols G through Z.

exp
A 16-bit hexadecimal value that will be assigned to the specified register or symbol.

Example:

```
AS $R1 -2 X 1408 $B7 X+15
```

This example causes -2 to be assigned to data register 1, 1408 to be assigned to the reserved symbol X, and 141D to be assigned to base register 7.

Clear All Directive

The clear all directive (C*) clears all defined breakpoints.

FORMAT:

C*

Change Memory Directive

The change memory directive (CH) changes the contents of a single specified memory location, or consecutive locations starting at that location, to specified value(s).

FORMAT:

CHΔexpΔrexp {Δrexp...}

ARGUMENT DESCRIPTIONS:

exp
First or only location whose contents will be changed.

rexp
Value(s) to be put in memory location(s).

Example 1:

```
CH 200 4FFF 1716
```

Execution of this directive puts the value 4FFF into location 200 and 1716 into location 201.

Example 2:

```
CH 100 0/10
```

In this example, locations 100 to 10F will be zero-filled.

Example 3:

```
CH 2000 0/10 1/10 2/10
```

This example shows how multiple repeat factors can be used: execution of this directive causes locations 2000 to 200F to be given a value of zero; locations 2010 to 201F to be given a value of 1, and locations 2020 to 202F to be filled with 2's.

Clear Directive

The clear directive (Cn) clears a specified breakpoint.

FORMAT:

Cn

ARGUMENT DESCRIPTION:

n
Number of the breakpoint; can be from 0 through 9.

Example:

C3

This directive causes breakpoint number 3 to be cleared.

Clear Bound Unit Directive

The clear bound unit breakpoint directive (CBn) clears a specified breakpoint for a bound unit.

FORMAT:

CBn

ARGUMENT DESCRIPTION:

n
Specifies the breakpoint to be cleared; must be a decimal digit from 0 to 9.

Example:

CB3

This directive causes breakpoint number 3 to be cleared for the bound unit previously defined by SB3.

Clear All Bound Unit Directive

The clear all bound unit directive (CB*) clears all bound unit breakpoints.

FORMAT:

CB*

Define Directive

The define directive (Dn) defines a specified directive line for future use and associates that line with a specified number. The directive line is stored on the file DEBUG.WORK and can be referred to by specifying in an execute (En) directive the number with which it was associated.

When you reuse a disk that has predefined directive lines from a previous execution, the lines may be referred to without redefining them. (See "Set Breakpoint Directive (Sn).") This prevents complex predefined directive lines from being respecified each time the system is reloaded for debugging the same problem.

FORMAT:

Dn△(directive line)

ARGUMENT DESCRIPTIONS:

n
Number with which the specified directive line is associated; must be from 0 through 9.

(directive line)
Directive line; can comprise a maximum of 126 characters.

Example 1:

```
D3 (CH 100 0)
```

This example associates the number 3 with the directive within the parentheses. Hereafter, each time the directive E3 (see "Execute Directive (En)" below) is executed, the parenthetical directive will be executed and location 100 will be zero-filled.

Example 2:

```
D4 ( )
```

By storing a null directive, this example deactivates a previously defined directive line 4 which no longer is required.

Display Memory Directive

The display memory directive (DH) causes one or more specified memory location(s) to be displayed in hexadecimal notation either on the operator terminal or on another specified device.

FORMAT:

DH {/lrn△} rexp {△rexp...}

ARGUMENT DESCRIPTIONS:

/lrn
Logical resource number of the device on which the information will be displayed.
Default: Current Debug output device.

rexp
Location(s) whose contents will be displayed. A minimum of one location may be displayed.

Example 1:

```
DH 200
```

Execution of this directive causes a typeout on the operator terminal of the contents of location 200.

Example 2:

```
DH/2 200/100
```

Execution of this directive displays the contents of location 200 to 2FF on the device associated with LRN2.

Dump Memory Directive

The dump memory directive (DP) causes a printout on the operator terminal or another specified device of an area of memory starting at a specified location. The printout comprises a minimum of eight locations, and is in hexadecimal and ASCII notations.

If the printout is written to a terminal and a value equal to or greater than 121 decimal was specified in the LL directive, 16 locations will be printed on each line.

NOTE: Up to 32K words of memory can be dumped in response to a single DP directive. Dumps of more than 32K must be performed as separate operations.

FORMAT:

DP {/lrn} Δrexp {Δrexp...}

ARGUMENT DESCRIPTIONS:

/lrn
Logical resource number of the device on which the display will occur.
Default: Current Debug output device.

rexp
Memory location(s) whose contents will be displayed. The display always is in a multiple of eight locations.

Example 1:

```
DP 200
```

Execution of this directive displays one line of memory in both hexadecimal and ASCII, starting at location 200.

Example 2:

```
DP/4 80/3C 200/240
```

This directive causes the contents of locations 80 to BB, and 200 to 43F to be displayed on the device associated with LRN 4. Although location 3C was specified in the directive, the display is through location BF because displays always are in multiples of eight locations.

Define Trace Directive

The define trace directive (DT) associates the directive line within the parentheses with the occurrence of a trace trap or a BRK instruction not already defined as a breakpoint. The specified directive line is stored in the file DEBUG.WORK for future use.

When you reuse a disk that has predefined directive lines from a previous execution, the lines may be referred to without redefining them. (See "Set Breakpoint Directive (Sn).")

FORMAT:

DTΔ(directive line)

ARGUMENT DESCRIPTION:

(directive line)
Directive line comprising maximum of 126 characters.

Example 1:

```
DT (AR)
```

This directive causes all registers to be displayed each time a trace trap occurs. (See "All Registers Directive (AR).")

Example 2:

```
DT ( )
```

This directive cancels usage of the predefined trace directive line.

Execute Directive

The execute directive (En) retrieves and executes a specified predefined directive line. This directive may not be embedded in define directive (Dn) lines; it is permitted in set breakpoint (Sn), define trace (DT), and print tract (PT) lines.

FORMAT:

En

ARGUMENT DESCRIPTION:

n
Number of the line to be executed; must be from 0 through 9.

Example 1:

```
D3 (CH  100  0)
E3
```

The directive E3 causes the retrieval and execution of line 3, which was previously defined in the define directive as CH 100 0.

Example 2:

```
D3 (CH  100  0)
S1  100  (E3)
```

In this example, the execute directive (E3) is embedded in a set breakpoint directive line. The execute directive will cause the retrieval and execution of line 3, which was previously defined in the define directive as CH 100 0.

End Trace Directive

The end trace directive (ET) disables the j-mode trace for a specific task on the next trap.

FORMAT:

ET LVL

ARGUMENT DESCRIPTION:

LVL
The Debug active level, as previously specified by an SL directive. LVL is preceded by one space. The trace must first have been enabled using the ST directive.

Redirect Debug Output Directive

The redirect debug output directive (FO) redirects output from the default debug user-in terminal to an alternate device, which must be either a printer or another KSR-compatible terminal. This directive allows messages, that result when a breakpoint or other condition occurs, to be sent to a device other than the terminal. It has no effect on input to debug.

FORMAT:

FO lrn

ARGUMENT DESCRIPTION:

lrn

Logical resource number associated with the printer or terminal to which Debug output is to be redirected. The lrn specified overrides any previously specified, and remains in effect until another FO directive is issued or until Debug is terminated. However, stored directive lines that include /lrn parameters take precedence over the FO directive. That is, the value specified for /lrn is used instead of the lrn specified in FO.

NOTE: There is no validation of the lrn specified. Thus, if an inappropriate device (e.g., diskette) is specified, no error message is issued to inform the user.

Example:

FO 2

Output is redirected from the terminal to the device associated with lrn 2.

GO Directive

The GO directive resumes execution on the current active level after a breakpoint and can optionally specify a limit-to-pause counter value which applies only to j-mode trace traps (see the Start j-mode Trace Directive).

FORMAT:

GO [∇LLLL]

ARGUMENT DESCRIPTION:

[∇LLLL]

Optionally specifies an ASCII expression of 1 to 4 hexadecimal digits greater than zero. The ASCII expression is preceded by one space.
Default: 1

Example:

S0 100 (DH 200/10;GO)

The task encountering breakpoint O will trap; the associated directive line will be executed by Debug and the last directive of the directive line (GO) will cause the task to be reactivated.

Conditional Execution Directive

The conditional execution directive (IF) allows a set of conditions to be tested prior to execution of other debug directives. The IF directive is intended to be used in a stored breakpoint directive line. It permits breakpoints to be reported without suspending the active level if the specified condition does not exist. When a breakpoint occurs for which an IF directive has been specified, the following actions occur:

- o Any directive preceding IF are executed
- o The IF conditions are evaluated, as follows:
  If TRUE, a line in the following format is displayed on the current debug output device

  "exp{^} {<, =, >}(,)hhhh..."

  and any directives following IF are executed. If a GO directive does not follow, the active level is suspended.
  If FALSE, no display occurs, and the directives following IF are not executed. The active level continues processing.

FORMAT:

IF exp{^} {<, =, >}(,)hhhh... ,

ARGUMENT DESCRIPTIONS:

exp
The memory address of a byte string argument. This must specify an address; $R cannot be used for exp. (No check for this error is performed however.)

{^} {<, =, >}

Specify the condition to be tested i.e., comparing the memory byte string value to the test parameter {^} optionally specifies logical negation; i.e., not less than, not equal, not greater than.

(,)
Indicates that the argument is right-byte aligned.

hhhhhh...
The test parameter, expressed in ASCII as a dense string of pairs of hexadecimal digits; each pair represents one byte. The test parameter may not be an assigned symbol (see assign directive). The length of the parameter is limited by the maximum size of a debug stored directive (127 bytes). The parameter's ASCII value must consist of pairs of hexadecimal values. If an odd number of hexadecimal values are specified, a command error is reported when the directive is executed and the task remains suspended to allow for correction.

{ ; , }

The IF directive terminator must be either a semicolon or a right parenthesis.

Example:

Assume that breakpoint 2, as defined below, is encountered, and that $B7 points to memory location 555F:

S2 135E (IF▽1000∧,3E;IF∨$B7=42D1;DP/5▽$B7/100;GO)

In this example two conditions must be true before the dump (DP) directive will be executed:

1. The rightmost byte at memory location 1000 must be less than or equal to 3E;
2. The byte string found at memory location 555F must be equal to 42D1.

If both conditions are met, the dump will be executed, and the active level will continue, in response to the GO directive. If either condition is not satisfied, the dump will not occur, and the active level will continue without suspension.

NOTE: The IF directive can be entered from the terminal, in which case its action corresponds to its entry in a stored directive line. However, using the IF directive from the terminal is of limited usefulness, since the conditions to be tested can be checked by using other directives (e.g., DH).

Print Header Line Directive

The print header line directive (Hn) causes a specified header line to be printed starting at the head of form or after a specified number of lines are skipped. The main uses of the print header line directive are to document printed information related to breakpoint or trace trap debugging, and to annotate a line printer memory dump.

FORMAT:

Hn {/lrn} Δ(headerΔ)

ARGUMENT DESCRIPTIONS:

n
Number of lines skipped before header line is printed; can be 1 through 9, or 0. 0 causes header to be printed at head of form.

/lrn
Logical resource number of device on which printout will occur.
Default: Current Debug output device.

(headerΔ)
Any ASCII characters and/or expressions; each expression must be preceded by a percent (%) sign. If a percent sign is to be printed, two percent signs must be used (%%). A header line must end with a space character; i.e., there must be a space immediately before the right parenthesis.

Example:

H0/2(DUMP OF BREAKPOINT FOR LEVEL %$SΔ)

This example illustrates a way to document dumps. As soon as a carriage return is typed, the above header will be printed at the top of a new page on the device identified by logical resource number 2.

List All Breakpoints Directive

The list all breakpoints directive (L*) causes a listing of all currently-defined breakpoints and their locations in memory. Stored directive lines, if any, are not displayed.

FORMAT:

$$L^*\{/lrn\}$$

ARGUMENT DESCRIPTION:

/lrn
Logical resource number of the device on which printout will occur.
Default: Current Debug output device.

Example:

```
L*
```

This directive causes a display of all breakpoints.

List Breakpoint Directive

The list breakpoint directive (Ln) causes the display of a particular breakpoint number that was set by a set breakpoint (Sn) directive, and its associated directive line.

FORMAT:

$$Ln\{/lrn\}$$

ARGUMENT DESCRIPTIONS:

n
Number of breakpoint whose directive line will be listed.

/lrn
Logical resource number of device on which printout will occur.
Default: Current Debug output device.

Example:

```
L2/4
```

This directive causes the display of the directive line of breakpoint 2 on the device associated with LRN 4.

List All Bound Unit Breakpoints Directive

The list all bound unit breakpoints directive (LB*) causes all currently active bound unit breakpoints to be displayed.

FORMAT:

$$LB^*\{(/lrn)\}$$

ARGUMENT DESCRIPTION:

/lrn
Logical resource number of the device on which the listing will occur.
Default: Current Debug output device.

Example:

```
LB*
```

All currently active bound unit breakpoints are listed at the current Debug output device.

List Bound Unit Breakpoint Directive

The list bound unit breakpoint directive (LBn) causes the directive line associated with a specified bound unit breakpoint to be displayed.

FORMAT:

$$LBn\left\{(/lrn)\right\}$$

ARGUMENT DESCRIPTIONS:

n
: The number of the bound unit breakpoint for which the directive line is to be listed.

/lrn
: Logical resource number of the device on which the directive line will be listed.
Default: Current Debug output device.

Example:

```
LB3/4
```

This directive results in the listing of the directive line associated with bound unit breakpoint 3. The listing occurs on the device associated with logical resource number 4.

Line Length Directive

The line length directive (LL) specifies the maximum line length of each line entered through the operator's terminal or another terminal in use.

FORMAT:

$$LL\Delta value$$

ARGUMENT DESCRIPTION:

value
: Number, in hexadecimal notation; must be between 1E (decimal 30) and 7E (decimal 126); e.g., 48, which is decimal 72.

Example:

```
LL 48
```

This directive signifies that the operator terminal or other terminal in use has a maximum line length of 72 decimal characters.

Print All Directive

The print all directive (P*) causes a printout of *all* lines predefined by Dn directives.

FORMAT:

P* {/lrn}

ARGUMENT DESCRIPTION:

/lrn
Logical resource number of device on which printout will occur.
Default: Current Debug output device.

Print Directive

The print directive (Pn) causes a printout of *specified* lines predefined by Dn directives.

FORMAT:

Pn {/lrn}

ARGUMENT DESCRIPTIONS:

n
Number of line to be printed; can be 0 through 9.

/lrn
Logical resource number of device on which printout will occur.
Default: Current Debut output device.

Print Trace Directive

The print trace directive (PT) causes a printout of a define trace directive line.

FORMAT:

PT { /lrn}

ARGUMENT DESCRIPTION:

/lrn
Logical resource number of device on which printout will occur.
Default: Current Debug output device.

Quit Directive

The quit directive (QT) clears breakpoints and disables the Debug trap handler before effectively aborting the Debug task group.

FORMAT:

QT

Reset File Directive

The reset file directive (RF) prohibits execution of directives that use the file DEBUG.WORK, until another specify file (SF) directive is issued. The directives that use DEBUG.WORK are: P*, Pn, PT, Sn, Hn, Dn, DT and SBn.

FORMAT:

RF

Set Breakpoint Directive

The set breakpoint directive (Sn) sets a numbered breakpoint at a specified location. When the breakpoint is encountered, the stored, specified directive line, if any, is executed; otherwise, there is a typeout indicating the contents of the location counter and the active priority level, and task execution is suspended. The "Set File" directive (SF), is a precondition for directive line execution.

If there is a preexisting directive line associated with a given breakpoint and that directive line is no longer applicable, specify in the define directive (Dn) a different directive line or clear the line by designating empty parentheses ( ).

The message format is:

($D) BPn $P=00xxxx $SL=00xx

$P=00xxxx
Designates location counter

$SL=00xx
Designates priority level

NOTES:

1. If a breakpoint is set in any of the following types of instructions, the breakpoint must be cleared (Cn directive) before continuing execution (GO directive): input/output, generic (BRK), scientific, LEV, invalid, or instruction with an illegal address syllable. You may avoid this restriction by clearing the existing breakpoint and then resetting it in a subsequent set breakpoint directive.
2. A GO directive embedded in an Sn directive line allows task execution to proceed after the desired operations have been performed, without further operator intervention.

FORMAT:

SnΔexp {Δ(directive line)}

ARGUMENT DESCRIPTIONS:

n
Number of breakpoint; can be 0 through 9.

exp
Location at which breakpoint will occur.

(directive line)
Directives that will be executed when breakpoint is encountered. Directive line can be a maximum of 126 characters.

Example 1:

```
S0 100 (DH 200/10;GO)
```

This directive will cause the display of locations 200 to 20F when location 100 is executed.

Example 2:

```
S0 100 ( )
```

This directive cancels any line previously associated with breakpoint 0.

Example 3:

```
S0 1000 (AR;C0;GO)
S1 1003 (S0 1000;GO)
```

The first directive line sets breakpoint number 0 at location 1000, causes a printout of all registers on the active level, and then clears breakpoint number 0 because the instruction at location 1000 is restricted (see Note 1 above).

The second directive line sets breakpoint number 1 at location 1003 and then reestablishes breakpoint 0 at location 1000; the second breakpoint line causes no visible action.

Set Bound Unit Breakpoint Directive

The set bound unit breakpoint directive (SBn) sets a numbered breakpoint for a specified bound unit or bound unit overlay. A given bound unit (BU) breakpoint refers to either roots or to overlays, but not to both. This directive informs the user that a bound unit or overlay has been loaded into memory, so that breakpoints can then be set at specified locations in the program. Because a bound unit is loaded at the time the task associated with it is created, the level number displayed when a BU breakpoint occurs is not necessarily the one used when requests for that task are later executed.

FORMAT:

$$\text{SBn}\Delta \begin{Bmatrix} \text{bound-unit-name} \\ \text{bound-unit-name/overlay-number} \\ \text{bound-unit-name/*} \\ * \\ */* \end{Bmatrix} \Delta(\text{directive line})$$

ARGUMENT DESCRIPTIONS:

n
: Breakpoint number; can be from 0 to 9.

bound-unit-name
: Name of the bound unit to which the breakpoint applies; up to six ASCII characters (first six characters of the file name).

overlay-number
: Hexadecimal number of the bound unit overlay.

*
: Stands for "all" roots or "all" overlays, depending on context.

(directive line)
: Directives to be executed when the bound unit/overlay is loaded; can be up to 127 characters long.

Example:

```
SB6  SOOZ/A  (IF 3D02=5354;VH $R1 -2;GO)
```

This directive sets breakpoint 6 for overlay number A_{16} of the bound unit named SOOZ. The directive line specifies that if the condition indicated is true (location 3D02 equals 5354), then the value of R1 is displayed. When overlay A is loaded into memory, its location is displayed on the operator terminal, and the directive line associated with breakpoint 6 is executed.

Specify File Directive

The specify file directive (SF) identifies the device on which the file DEBUG.WORK is located. Since the function of the SF directive is to find the work file, it should be the first directive executed; failure to do this results in the issuing of an error message as soon as a directive that requires the work file is used. Debug accesses the work file by using physical I/O, bypassing the file system.

FORMAT:

SF△lrn

ARGUMENT DESCRIPTION:

lrn
Logical resource number of the disk device on which the file DEBUG.WORK is located; must be specified in hexadecimal notation.

Example:

```
SF B
```

This example specifies that the work file is on the device whose logical resource number is 11 (decimal).

Set Level Directive

The set level directive (SL) sets the active priority level to a specified value. This level remains in effect until another SL directive is issued. The level may be temporarily changed via the set temporary level directive (TL).

FORMAT:

SL△exp

ARGUMENT DESCRIPTION:

exp
Number of active priority level.
Default: 0

Example 1:

```
SL C
```

This directive sets the active priority level to 12 (decimal). If the AR directive is entered after the above SL directive, the registers on level 12 are displayed.

Example 2:

This example designates how the active level can be designated, permanently changed, and temporarily changed.

```
SL C        The active level is 12 (decimal)
 :
SL A        The active level is 10 (decimal)
 :
TL B;AR     The active level temporarily is 11 (decimal)
```

After the desired action(s) are performed, the active level reverts to level 10 (the level specified in the last SL directive).

Start j-mode Trace Directive

The start j-mode trace directive (ST) sets the given task's M1 register j-bit on. As a result, any departure from the current processing sequence will cause a trap. Debug treats the trap as a "trace trap." The following points apply:

- o j-mode trace can only be started for a task which is currently suspended due to a true breakpoint.
- o The "start j-mode trace" directive will be refused if the task is suspended due to a bound unit breakpoint.
- o j-mode processing is specific to a given task and is shut off or restored at the monitor call interfaces.
- o When a task is running in j-mode, debug's handling of successive traps is governed by the "limit-to-pause" counter of the GO directive.
- o Limit-to-pause has a default value of 1, but may be set to an arbitrary value via the GO directive. Debug decrements the limit-to-pause once for each occurrence of a trace trap. When limit-to-pause assumes the value zero, the trapped task is suspended to permit operator action. When the task is reactivated (GO[LLLL]) the limit-to-pause is reset to the default value or to a user-specified value.

FORMAT:

ST LVL

ARGUMENT DESCRIPTION:

LVL
The Debug activity level. The SL directive must first specify Debug's active level. This in turn must correspond to the level of the task in question.

Set Temporary Level Directive

The set temporary level directive (TL) sets the active priority level to a *temporary,* specified value. The level specified in the TL directive remains in effect until an SL or another TL directive is issued, or until the end of the directive line. If the end of the line is reached before another SL or TL directive is encountered, the value specified in the last SL directive becomes the active priority level.

FORMAT:

TLΔexp

ARGUMENT DESCRIPTION:

exp
Value designating the temporarily active priority level.

Example:

```
SL 20
TL A;AR
TL B;AR
```

The first TL directive designates level 10 as the temporarily active priority level so that all registers on that level can be displayed via the subsequent AR directive.

The second TL directive designates level 11 as the temporarily active priority level so that all registers on that level can be displayed via the subsequent AR directive.

After the last TL directive is executed, the active level will be 20 (the level specified in the last set level directive (SL)).

Print Hexadecimal Value Directive

The print hexadecimal value directive (VH) causes a printout, in hexadecimal, of the value of each specified expression.

FORMAT:

VH {/lrn} Δexp {Δexp...}

ARGUMENT DESCRIPTIONS:

/lrn
Logical resource number of device on which printout will occur.
Default: Current Debug output device.

exp
Location whose value will be displayed; if more than one expression is specified, the display will be of the value resulting from the computation of the expressions. A value is always a 16-bit quantity, not an address, so VH cannot be used for LAF address computations.

Example:

```
VH .+100-M
```

This directive causes the display of the result of the computation defined by the last referenced memory location plus 100 (hexadecimal) minus the value assigned to the temporary symbol M.

EXAMPLES ILLUSTRATING USAGE OF DEBUG DIRECTIVES

The following examples contain typical operations that may be performed using the Debug program.

Example 1:

1. Establish header.

   ```
   HO (DUMP &M OF AREA 0)
   ```

   The header will be printed on the current debug output device.

2. Predefine a directive line.

   DO (HO/3;AR/3;DH/3 20/4 [8A] /1A [8B] /1A Y/100)

   There will be displays of the following information:
   Header (requested by HO/3)
   Registers (requested by AR/3)
   Specified memory locations:
   Four locations beginning at location 20; designated by 20/4
   ISA information for level 10 (designated by [8A]), level 11 (designated by [8B]), and 256 locations beginning at the location assigned to variable Y.

3. Initialize header variable to 0.

   AS M 0

4. Set breakpoints in code being tested.

   SO 300 (AS M M+1)
   S1 4A6 (AS M M+1)

5. Set active level to 18_{16}; start j-mode trace; pause after 12_{16} traps.* SL 18; ST 18; GO 12.

6. When the breakpoint occurs, execute predefined directive line 0 and then continue.

   E0
   GO

Example 2:

1. Set Breakpoint at the entry point of a subroutine used by multiple application programs.

   S0 300 ( )

2. Set the next breakpoint to the address that contains the instruction that should be executed immediately after the subroutine. The address should appear in register $B5.

   S1 $B5 ( ); GO

3. When the first breakpoint occurs, display registers.

   AR

Example 3:

Proceed from HALT instruction.

AS $P $P+1

DEBUGGING PROGRAMS WITHOUT USING DEBUG

If there is not enough memory for Debug, there are several ways of monitoring the system to verify proper sequencing of memory and/or register contents within routines, or proper task sequencing, without using Debug. Each method requires manual insertion of monitoring code, and implies that space exists within the system for it.

There are two ways of creating space to insert monitor points:

- o Leave space temporarily in various application units
- o Append monitoring points using Patch. (Patch is described in Section 7.)

The sophistication of the monitoring performed depends on the stage of the application development and testing. The monitoring routine can be a simple halt instruction, a Debug breakpoint, or your own routine. In each case, you may construct what is required at the time.

When single-word halt instructions are used as monitoring points, the use of the D0 single instruction capability is convenient. The instruction word replaced by the halt may be entered into the instruction register (D0) when the halt occurs, even if the full instruction occupies more than one word in memory. However, the halt must replace the first word of the instruction.

Example:

Source Line	Object Text	
LAB $B2,$B4,TAG	loc/ABC4	instruction word
	loc+1/0004	tag value

ABC4 may be replaced by a halt (all zeros). When the P-register (E0) halts at loc+1, the instruction register (D0) may be changed back to ABC4 and execution continued. Since this does not change the content of loc, the halt will reoccur the next time the code sequence at loc is executed.

If space is allocated in application units, it may be used by invoking Debug or Patch.

Deactivating Real-Time Clock

Some applications require that the real-time clock (RTC) be activated at load time. While the clock is turned on, the CPU is difficult to use in single instruction mode because the RTC is continually generating an interrupt at clock level 4. In the early stages of application debugging, it may be useful to turn off the clock to facilitate "stepping" through a code sequence without interference. This is easily done using the capability of executing a single instruction from the D0 register, as described in the following procedure.

To turn off the clock, use the capability of executing one instruction from the instruction register (whose selection code is D0) to execute an RTCF instruction while in single instruction mode. Perform the following steps:

1. Press Stop, and record value in E0
2. Select D0; change to 0005
3. Press Execute (this turns off clock)
4. Select D0; change to 0000
5. Select E0; change to recorded E0 halt address
6. Press Ready and Execute

After "stepping" through a code sequence, repeat this procedure using an RTCN instruction. Some system functions require that the clock be on. (Specify 0004 for step 2.)

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.

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.

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.

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.

DUMP EDIT UTILITY PROGRAM

The Dump Edit utility program produces, on the user output file, a dump of some or all of the contents of a file that contains a dump resulting from a previous execution of the MDUMP utility program, or a dump of main memory so that you can determine the configuration under which Dump Edit is executing.[1]

[1] It is recommended that the user output file be a suitable file for the amount of memory that will be dumped; the user output file must be capable of receiving a 132-character line.

Dump Edit produces a logical (edited format) and a physical (image format) dump; if desired, you can request in the DPEDIT command (described below) that only one of those dumps be produced. A logical dump comprises, in edited format, the location and contents of hardware-dedicated memory locations, the system control block, memory pool definitions, each group control block (GCB), and each task control block (TCB), the work space blocks associated with each GCB and the indirect request blocks (IRBs) and trap save areas (TSAs) associated with each TCB. Logical and physical dumps of a disk file are printed in both hexadecimal and ASCII notation, with duplicate lines indicated as suppressed.

DPEDIT processing can be stopped at any time by depressing the "BREAK" key. A "QUIT" message appears on the user's terminal when the processing stops. At this point, the start command (SR) unwind command (UW), the program interrupt command (PI) or the NEW-PROC command may be specified. The end-of-processing details are automatically handled and control returns to the ECL processor (command level) with a successful sub-task completion status.

Dump Edit error messages are described in the *System Messages* manual and summarized in Table 6-2.

NOTE: 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., they both must be either short-address form (SAF) or long-address form (LAF).

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.

Operating Procedure for Dump Edit

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. Load Dump Edit by specifying the DPEDIT command, described below.

DPEDIT Command

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

FORMAT:

DPEDIT [path] [ctl_arg]

ARGUMENT DESCRIPTIONS:

path
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'}
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 address.

Default: High memory address of the dump file.

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

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
```

This command invokes DPEDIT and is requesting a logical dump of only the System and Debugger groups from current main storage.

Example 4:

```
DPEDIT -MEM -GROUP AB -NP
```

Assume there is no task group whose name is AB. This command invokes DPEDIT and is requesting a very abbreviated logical dump consisting of only Hardware Dedicated Locations, Memory Pool Data, and System Control Block.

Interpreting Dump Edit Dumps

Dump Edit Line Format

The format of each Dump Edit line is as follows:

Print position(s):

3 through 6
Four hexadecimal digits designating the starting address of the line of dump information; the hexadecimal digit in print position 6 always is 0.

7
Slash (/).

8 through 10
Blanks.

11 through 91
Contents of 16 consecutive words; each word is represented by four hexadecimal digits and is followed by a blank space.

92 through 95
Blanks.

96 through 127
ASCII representation of the previous group of 16 consecutive words. If a byte is not an ASCII character, a period (.) is designated.

128 through 132
Blanks.

If a *duplicate line* is encountered, it is indicated as follows:

Print positions:

1 through 11	Blanks
12 through 93	* * * * * * * * * * * *
94 through 132	Blanks

Logical Dump Format

Figure 6-1 illustrates the format of logical dumps produced by Dump Edit.

Table 6-3 describes supplemental information that may occur in logical dumps. This information occurs at the locations where applicable. (The dump and supplemental information are written to the user output file.)

Figure 6-2 contains a sample logical dump, which was requested by the command:

```
DPEDIT -MEMORY -TO X'2000'
```

```
MAIN STORAGE DUMP  year/month/day  hour:minute.tenths-of-minute  DUMPEDT-nnnn-mm/dd/hhmm
         0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
HARDWARE DEDICATED LOCATIONS
one or more lines in the "standard" Dump Edit format (see "Dump Edit Line Format" above).
SYSTEM TIME OF DUMP  year/month/day  hour:minute:second
SYSTEM CONTROL BLOCK    nnnn[a]  (where nnnn are four hexadecimal digits for SAF and
                                  five for LAF, designating the starting address of the
                                  named block)
one or more lines in the "standard" Dump Edit format (see "Dump Edit Line Format" above).
     :
     :
MEMORY POOL DEFINITIONS
POOL_NAME  START_ADDRESS  END_ADDRESS  SIZE (WORDS)  UNUSED(WORDS)  NUMBER_OF_USERS
    ID         nnnn          nnnn         nnnn          nnnn             nnnn
     :          :             :            :             :                :
     :          :             :            :             :                :
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
GROUP CONTROL BLOCK     nnnn[a]  (where nnnn are four (SAF) or five (LAF) hexadecimal
TASK CONTROL BLOCK               digits designating the starting address of the named
WORK SPACE BLOCK                 block)
one or more lines in the "standard" Dump Edit format (see "Dump Edit Line Format" above).
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -[b]
SYSTEM CONTROL BLOCK    nnnn[a]  (where nnnn are four hexadecimal digits for SAF and
                                  five for LAF, designating the starting address of the
                                  named block)
one or more lines in the "standard" Dump Edit format (see "Dump Edit Line Format" above).
     :
     :
MEMORY POOL DEFINITIONS
POOL_NAME  START_ADDRESS  END_ADDRESS  SIZE (WORDS)  UNUSED(WORDS)  NUMBER_OF_USERS
    ID         nnnn          nnnn         nnnn          nnnn             nnnn
     :          :             :            :             :                :
     :          :             :            :             :                :
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
GROUP CONTROL BLOCK     nnnn[a]  [bound-unit-name][c]  [GG][d]  (where nnnn are four (SAF) or
TASK CONTROL BLOCK               five (LAF) hexadecimal digits designating the starting
IND REQUEST BLOCK                address of the named block and GG designates the group id)
TRAP SAVE AREA
WORK SPACE BLOCK
one or more lines in the "standard" Dump Edit format (see "Dump Edit Line Format" above).
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -b
     :                        :
     :                        :
DUMP COMPLETE
```

[a]There can be multiple occurrences of this line for each task group.

[b]Dashes indicate that subsequent information is for another task group.

[c]For TASK CONTROL BLOCK, nnnn is followed by the first six characters of the BU name associated with the TCB.

[d]For GROUP CONTROL BLOCK, nnnn is followed by a two-character group id.

Figure 6-1. Format of Logical Dumps Produced by Dump Edit

```
MAIN STORAGE DUMP     1977/11/28 1142:10.3 a      DUMPEDIT-0100-11/09/0943          GCOS6 MOD400-L100-11/21/0810          PAGE 0001

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

HARDWARE DEDICATED LOCATIONS
 00000/   0000 0000 0000 0000 0000 0000 0000 0000   0000 0000 0000 0000 0000 0000 0000 0000    ................................
 00010/   0000 44B2 0000 0000 000C 0006 0004 0000   0000 01AB 0000 8380 0000 1D9C 0000 0000    ..D.............................
 00020/   0000 0100 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 585D   0000 0155 0000 51AB 0000 0159 0000 001B    ...M...O...Q..X]...U..Q....Y....
 00080/   0000 0000 0000 0000 0000 019E 0000 38ED   0000 0433 0000 35E3 0000 06F9 0000 0000    ..............8....3..5.........
 00090/   0000 0000 0000 3711 0000 0000 0000 0000   0000 3B9C 0000 3C76 0000 3D50 0000 3E2A    ......7...........;...<V..=P..>*
 000A0/   0000 0000 0000 0000 0000 38ED 0000 3A28   0000 0000 0000 0000 0000 0000 0000 88DC    ..........8...:(................
 000B0/   0000 0000 0000 0000 0000 0000 0000 0000   0000 0000 0000 0000 0000 0000 0000 0000    ................................
            *    *    *    *    *    *    *    *     *    *    *    *    *    *    *    *b
 00100/   0FFE 8AD3 0F02 8AD3 0F02 8AD3 0F02 8AD3   0F02 8AD3 0F02 8AD3 0F02 8AD3 0F02 8AD3    ................................

SYSTEM TIME OF DUMP       1977/11/28 1142:12.5


SYSTEM CONTROL BLOCK    001AB
 001A0/   0000 0000 01A4 4000 0000 0FFF 3131 2F32   312F 3038 3130 0000 45DD 0000 0000 0000    ......d.....11/21/0810..E.......
 001B0/   09BB 0000 7963 0000 0000 0000 0996 0000   5A03 0000 09BB 0000 7843 0015 222A 0000    ....YC..........Z........XC..*"..
 001C0/   5AFD 0235 158D BE08 0064 0000 0000 020A   0000 0000 0000 5A29 0001 FFFF 0068 0035    Z..5.....D............Z).....H.5
 001D0/   0000 6FE3 0000 0996 0000 0303 0000 0AB4   0015 FE4F 3000 0000 0000 0001 0000 0000    ..O.................OO..........
 001E0/   0000 0000 0000 0002 0000 0000 0000 2442   FFFE 0523 007A 0067 0000 1A24 0000 0336    ..............$B...#.Z.G...$...6
 001F0/   0000 1DCD 0001 FFFF 0000 0000 0002 0000   0000 0000 0000 0000 0000 0000 0000 0000    ................................


MEMORY POOL DEFINITIONS
  POOL_NAME      START_ADDRESS    END_ADDRESS    SIZE(WORDS)    UNUSED(WORDS)    NUMBER_OF_USERS
     $$              05A20           0795F          01F40           01A00              00001
     AB              07960           1FFFF          186A0           17260              00001

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - c
```

Figure 6-2. Sample Logical Memory Dump

```
MAIN STORAGE DUMP     1977/11/28 1142:10.3      DUMPEDIT-0100-11/09/0943          GCOS6 MOD400-L100-11/21/0810          PAGE 0002

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

 GROUP CONTROL BLOCK     07963        $H
07960/    0004 0000 0000 0000 0378 2448 8067 0000  0000 0000 9718 0000 0000 0000 5140 0000   .........X$H.G..............Q@..
07970/    0000 79B2 0000 7999 0000 8BA1 0000 7AE3  0000 0000 0000 88DC 0000 098B 0001 FED3   ..Y...Y.......Z.................
07980/    0001 FE93 0000 0000 0000 0000 0000 0016  0000 0000 0000 0000 0002 0000 0000 0000   ................................
07990/    7A1C 0001 0000 0000 0000 0000 0000 0002  000A 0000 0000 0000 0000 0000 0000 0000   Z...............................

  TASK CONTROL BLOCK     08BDC        UPEDIT
08BA0/    0004 0000 7BA1 0000 88DC 0017 0000 0000  0000 4000 0000 0000 4000 0000 7BEC 00C3   ................@.....@.......
08BB0/    0000 7BD6 0000 0000 7BD6 0000 3002 0000  0017 0000 01AB 0001 FE93 0000 0A55 0001   ............U..............U..
08BC0/    FF13 0001 FF13 0001 FF13 0000 0000 8CU4  0000 7BB2 0000 7BB2 0000 0000 0000 0224   ...............................$
08BD0/    0000 0224 0000 7963 0000 7A1C 0000 88DC  0000 0000 0000 0000 0000 FFFF 0303 0000   ...$...YC..Z....................
08BE0/    1C2E 6003 0000 44E4 0000 01AB 0000 0000  0000 7434 0000 38ED 0000 58C7 0000 458F   ..@...D.............I4..8...8...E.
08BF0/    0000 0005 0000 0084 0001 002D 002B 0012  FF00 FF00 FF00 FF03 FF20 FF00 FF00 0000   ............-.+..... ..........
08C00/    0000 0000 0000 0000 0000 0000 0000 0000  0000 0000 0000 0000 0000 0000 0000 0000   ................................
08C10/    0000 0000 0000 0000 0000 0000 0000 0000  0000 0000 0000 0000 0000 0000 0000 0000   ................................

  IND REQUEST BLOCK     00224
00220/    0000 0217 0000 0204 0000 0000 0000 7B63  0000 7963 0000 45A3 000B 0000 88DC 0000   ..............C..YC..E.........
00230/    0000 0000 023E 0000 0000 0000 0000 0000  0000 0000 0000 0000 0000 0000 0000 024B   .....>.........................K

 TASK CONTROL BLOCK     07A1C        EC
07E00/    0004 0000 0000 0000 0000 0000 0000 0000  0000 0000 0000 0000 0000 0000 0000 0067   ...............................G
07F00/    0000 5B42 0000 0000 5B42 0000 30B6 0000  0016 0000 7AE3 0000 45F7 0000 7B87 0001   ..[B....[B..0.....Z...E.{.....
07A00/    FF13 0001 FF13 0001 FF13 0000 0000 0000  0000 5AF0 0000 5AF0 0000 0000 0000 45C3   ..............Z...Z.....{...E.
07A10/    0000 45C3 0000 7963 0000 0000 0000 7A1C  0000 0000 0000 43E2 0000 FFFF 0303 0000   ..E...YC.....Z.....L...........
07A20/    12B4 6003 0000 7594 0000 01AB 0000 14C5  0000 45A5 0000 0000 0000 0080 0000 45D0   ..@...U.....E...........E.....
07A30/    0000 007D 0017 0089 0001 0000 FFFF 0016  FF00 FF00 FF00 FF03 FF20 FF00 FF03 0000   .....'.......... ..............
07A40/    0000 0000 0000 0000 0000 0000 0000 0000  0000 0000 0000 0000 0000 0000 0000 0000   ................................
07A50/    0000 0000 0000 0000 0000 0000 0000 0000  0000 0000 0000 0000 0000 0000 0000 0000   ................................

  IND REQUEST BLOCK     045C3
045C0/    38ED 0000 0000 0000 0000 0000 7A63 0000  7963 0000 0000 0002 0000 7A1C 0000 0000   8.............ZC..YC........Z.....
045D0/    0000 0000 0000 0000 0000 7963 0000 7A1C  0021 0000 45D0 0000 0000 0000 45B2 0000   ..........YC..Z..!..E.....E.....

    TRAP SAVE AREA     043E2
043E0/    1DC1 0000 0000 0000 3F02 0001 0001 80B1  0000 5BF3 0000 5BF5 0000 7B09 0000 74A3   .........?.........[...[.....T.
043F0/    0000 7A97 0000 0000 0000 13CF 0000 7BU3  0000 4450 FF00 0000 0000 0000 0000 0000   ..Z.............DP..............
04400/    0000 0000 0000 0000 0000 0000 0000 0000  0000 0000 0000 19FD 0000 1FE8 0000 20B4   ............................. .
04410/    0000 0EE2 0000 01AB 0000 7B63 0000 88DC  0000 88A5 0000 1300 0024 007D 0017 FF00   ...........C.............$.....
04420/    0002 0000 0303 0000 0000 500E 0000 159E  0000 7B63 0000 7BU6 0017 0027 0005 00E0   ..........P.....C.....'.....'....
04430/    0008 0000 5046 0000 7B09 7BU9 0000 0000  7B63 208C 0000 0000 7B82 0017 FFFF 0017   ....PF..........C ..............
04440/    0004 007F 0000 0000 1DBD 0000 0001 0000  1DBU 0000 0000 437A 3F24 0025 0001 80C1   ....................CZ?$.%....
  ⋮

 WORK   SPACE BLOCK     08BA1
08BA0/    0004 0000 7BA1 0000 88DC 0017 0000 0000  0000 4000 0000 0000 4000 0000 7BEC 00C3   ................@.....@.......
08BB0/    0000 7BD6 0000 0000 7BD6 0000 3002 0000  0017 0000 01AB 0001 FE93 0000 0A55 0001   ............U..............U..
08BC0/    FF13 0001 FF13 0001 FF13 0000 0000 8CU4  0000 7BB2 0000 7BB2 0000 0000 0000 0224   ...............................$
08BD0/    0000 0224 0000 7963 0000 7A1C 0000 88DC  0000 0000 0000 0000 0000 FFFF 0303 0000   ...$...YC..Z....................
08BE0/    1C2E 6003 0000 44E4 0000 01AB 0000 0000  0000 7434 0000 38ED 0000 58C7 0000 458F   ..@...D.............I4..8...8...E.
08BF0/    0000 0005 0000 0084 0001 002D 002B 0012  FF00 FF00 FF00 FF03 FF20 FF00 FF00 0000   ............-.+..... ..........
```

[a]Indicates in the format hhmm:ss.s, the hour, minute, and second at which the dump was taken.

[b]Each duplicate line is indicated by a row of asterisks * * * *.

[c]A dashed line - - - - - - indicates the end of information for the above group and the beginning of information for the next group.

Figure 6-2. Sample Logical Memory Dump (Cont.)

TABLE 6-3. SUPPLEMENTAL INFORMATION THAT MAY OCCUR IN LOGICAL DUMPS PRODUCED BY DUMP EDIT

Information	Meaning
ADDRESS POINTER IS INVALID (LAF mode only)	An address contained in the dump file is invalid for LAF mode.
DATA NOT READABLE	Dump Edit tried to read the contents of a non-existent location on the dump file, or an uncorrectable read error was encountered.
DUPLICATE GCB STARTING ADDRESS	The specified group control block starting address has already been displayed.
DUPLICATE TCB STARTING ADDRESS	The specified task control block starting address has already been displayed.
DUPLICATE WORK SPACE BLOCK	The specified work space block starting address has already been displayed.
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 GCBS EXCEEDS 40	There are more than 40 group control blocks.
NUMBER OF TCBS EXCEEDS 40	There are more than 40 task control blocks for the current group control block.
NUMBER OF IRBS EXCEEDS 25	There are more than 25 indirect request blocks for the current task control block.
NUMBER OF TSAS EXCEEDS 10	There are more than 10 Trap save areas for the current task control block.

Physical Dump Format

The format of physical dumps is illustrated below:

DUMPEDIT year/month/day hour:minute.seconds

MAIN STORAGE DUMP GCOS6/MDTS110 Page nnnn

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

one or more lines in the "standard" Dump Edit format (see "Dump Edit Line Format" above).

Table 6-4 describes supplemental information that may occur in physical dumps produced by Dump Edit. This information occurs at the location(s) where applicable. (The dump and supplemental information are written to the user output file.)

Figure 6-3 illustrates a sample physical dump produced by Dump Edit which was requested by the command:

DPEDIT -MEMORY -TO X'2000'

TABLE 6-4. SUPPLEMENTAL INFORMATION THAT MAY OCCUR IN PHYSICAL DUMPS PRODUCED BY DUMP EDIT

Information	Meaning
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. This message is preceded by the line's starting address.

```
MAIN STORAGE DUMP     1977/11/28 1142:10.3      DUMPEDIT-0100-11/09/0943          GCOS6 MOD400-L100-11/21/0810          PAGE 0014

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

00000/ 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000  ................................
00010/ 0000 44B2 0000 0000 000C 0004 0004 0000 0000 01AB 0000 8380 0000 1D9C 0000 0000  ..D.............................
00020/ 0000 0100 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 585D 0000 0155 0000 51AB 0000 0159 0000 001B  ...M...O...Q..X]...U..Q....Y....
00080/ 0000 0000 0000 0000 0000 019E 0000 38ED 0000 0433 0000 35E3 0000 06F9 0000 0000  ..............8....3..5.........
00090/ 0000 0000 0000 3711 0000 0000 0000 0000 0000 3B9C 0000 3C76 0000 3D50 0000 3E2A  ......7.........;...<V..=P..>*
000A0/ 0000 0000 0000 0000 0000 38ED 0000 3A26 0000 0000 0000 0000 0000 0000 0000 88DC  ..........8...:(................
000B0/ 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000  ................................a
            *    *    *    *    *    *    *    *    *    *    *    *    *    *    *    *
0010/ 0FFE 8AD3 0F02 8AD3 0F02 8AD3 0F02 8AD3 0F02 8AD3 0F02 8AD3 0F02 8AD3 0F02 8AD3  ................................
0011/ 0F02 8AD3 0F02 8AD3 0F02 8AD3 0F02 8AD3 0F02 8AD3 0F02 8AD3 0F02 8AD3 0F02 8AD3  ................................
            *    *    *    *    *    *    *    *    *    *    *    *    *    *    *    *
0015/ 0F02 8AD3 0F02 8AD3 0F02 8AD3 0F02 8AD3 0F02 8AD3 0F02 8AD3 8F45 0008 EFEF 9070  ........................C.....P
0016/ FF00 8243 FFFD 8DC3 FFFA 099B D8F8 FFCD 8285 0597 DCF8 FFD1 8DD5 0913 DFC3 001C  ...C............................
0017/ 8FC3 0008 EFEF 9070 8000 82C3 FFFF 0080 0503 8BC0 0001 8FC3 FFFC 0080 83CB 001C  .......P........................
0018/ F870 0300 F453 C857 3C03 6C00 F8C3 0061 D3C0 1961 0001 0F00 8878 FFDA 2000 A854  .P...S.W<.L....A...A.....X.. ..T
0019/ 0001 0103 8F76 FFDD 4080 9878 FFCC 0001 0000 0001 0C08 8385 0000 0000 0000 0000  .....X..@..X....................
001A/ 0000 0000 01A4 4000 0000 0FFF 3131 2F32 312F 3038 3135 0000 458F 0000 0000 0000  ......@.....11/21/0810..E.......
001B/ 098B 0000 7963 0000 0000 0000 0996 0000 5A03 0000 098B 0000 7843 0015 222A 0000  ....YC..........Z.......xC..*...
001C/ 5AFD 0235 158E B678 0064 0000 0000 020A 0000 0000 0000 5A29 0001 FFFF 0068 0035  Z..5...X.D.............Z).....H.5
001D/ 0000 6FE3 0000 0996 0000 0303 0000 0AB4 0015 FE4F 3000 0000 0000 0001 0000 0000  ..O.................00..........
001E/ 0000 0000 0000 0002 0000 0000 0000 2442 FFFE 0768 0078 0067 0000 1A24 0000 0336  ..............$H...K.Z.G...$...6
001F/ 0000 1DCD 0001 FFFF 0000 0000 0002 0000 0000 0000 0000 0000 0000 0000 0000 0000  ................................
0020/ 0000 0000 0378 2456 FFFF 0000 0000 0217 0000 0000 0000 0000 0000 0FBD 0000 0378  .....X$V.......................X
0021/ 0000 0FC3 8047 0235 158E BC54 0000 0000 0000 0000 0000 0327 0000 0378 0000 1E4E 0027  .....G.5...T.........'...X...N.'
0022/ 0000 0217 0000 02D4 0000 0000 0000 7863 0000 7963 0000 45A3 000B 0000 88DC 0000  ..............C..YC...E.........
0023/ 0000 0000 023E 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 024B  .....>.........................K
0024/ 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0258 0000 0000 0000  ....................X...........
0025/ 0000 0000 0000 0000 0000 0000 0000 0000 0000 0265 0000 0000 0000 0000 0000 0000  ....................F...........
0026/ 0000 0000 0000 0000 0000 0000 0272 0000 0000 0000 0000 0000 0000 0000 0000 0000  ............R...................
0027/ 0000 0000 0000 027F 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000  ................................
0028/ 028C 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0299 0000 0000  ................................
0029/ 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 02A6 0000 0000 0000 0000 0000  ................................
002A/ 0000 0000 0000 0000 0000 0000 0000 02B3 0000 0000 0000 0000 0000 0000 0000 0000  ................................
002B/ 0000 0000 0000 0000 02C0 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000  ................................
002C/ 0000 02CD 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 02DA 0000  ................................
002D/ 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 02E7 0000 0000 0000 0000  ................................
002E/ 0000 0000 0000 0000 0000 0000 0000 0000 02F4 0000 0000 0000 0000 0000 0000 0000  ................................
002F/ 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000  ................................
0030/ 0000 9971 0285 9C91 8DD1 0902 8381 9870 0811 8385 D851 8753 83C0 1D91 D851 8753  ...Q.........P.....Q.S.....Q.S
0031/ 83C0 1D8F 8854 83C0 1D7A 8854 83C0 0975 8854 83C0 0970 8854 83C0 09F4 0000 5A14  .....T...Z.T...U.T...P.T......Z.
0032/ 412A 2A42 5245 4148 2A2A 0000 1E4E 0000 0217 00C5 FF01 0000 0204 0000 01AB 0001  A**BREAK**...N..................
0033/ 0000 0000 0000 0000 0000 0000 0001 0000 0000 0000 0001 0000 0000 0000 0000 0000  ................................
0034/ 0000 4586 0001 0001 0000 0354 0025 0050 0000 0000 0000 4586 0001 0001 0000 78C3  ..E........T.#.P......E.......x.
0035/ 000B 0010 0000 0000 4144 5045 4449 543A 2028 3032 3031 3035 2920 3137 2031 3338  ........ADPEDIT: (0201105) 17 138
0360/ 3020 3130 3030 2030 3030 3020 2020 2020 2020 2020 2020 2020 2020 2020 2020 2020  0 1000 0000
0037/ 2020 2020 2020 2020 2020 2020 2020 2020 0000 0000 2453 A031 B0A8 AD70 0000 C7A7  ....$S.1...P....
0038/ CAC1 0000 0000 AD5E 0000 0000 03B7 0000 59ED 0000 74A1 0000 0000 0000 0000 0000  .......>........Y...T...........
0039/ 3E2A 0000 098B 0000 7813 0000 7813 0000 0000 0000 0000 0000 0000 0002 0000 0000  >*......X...X...................
003A/ 01FD 0000 0000 FFFF 0000 0000 0009 0000 0000 0000 0000 0000 01CC 0000 0000 0000  ................................
0038/ 0000 0000 0000 0000 0000 0000 000F 0000 0000 0000 0000 0000 0000 0000 0000 0000  ................................
003C/ 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000  ................................
  ⋮
```

[a]A row of asterisks (****) indicates that the preceding line is repeated.

Figure 6-3. Sample Physical Memory Dump

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.

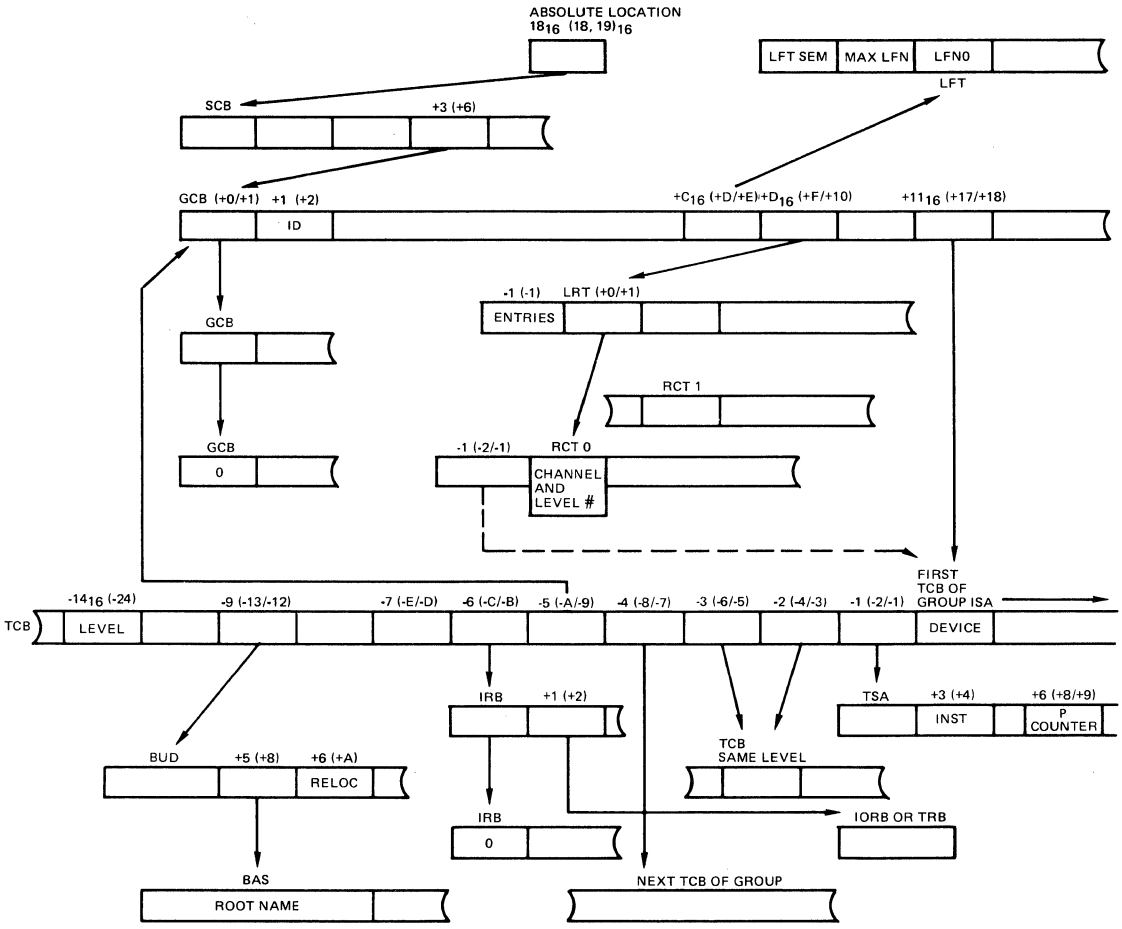


Figure A-1. Data Structure Map

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+C (+D/+E)
Pointer to LFN0 of logical file table (LRT).

GCB+D (+F/+10)
Pointer to task group's logical resource table (LRT).

GCB+11 (+17/+18)
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/priority level of the resource (i.e., input/output device or task).

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-14 (-24)
Hardware-assigned priority level of the task.

TCB-9 (-13/-12)
Pointer to current bound unit BUD.

TCB-7 (-E/-D)
Pointer to end of queue of requests for the task.

TCB-6 (-C/-B)
Pointer to start of queue of requests for the task (e.g., I/O requests for a driver).

TCB-5 (-A/-9)
Pointer to the group control block (GCB) for the group to which this task belongs.

TCB-4 (-8/-7)
Link to the queue of this group's TCB's.

TCB-3 (-6/-5)
Pointer to last TCB on that priority level.

TCB-2 (-4/-3)
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 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-1A (-2C/2B) 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.

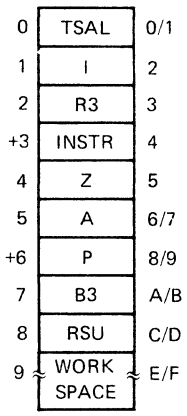
User Level Active at the Time of Dump

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, examine the contents of the instruction/P address entries. An 0001 instruction (monitor call) indicates that your program called upon a system service whose function code can usually be found at the P address-1 in a work space block of the task group encompassing the start address of your bound unit (otherwise look for this address in the physical dump portion of DPEDIT output). 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+11 (+17/+18) to determine the location of the first task control block (TCB) of the task group.
4. Go to location TCB-9 (-13/-12) to determine the location of your current bound unit descriptor (BUD).
5. Go to location BUD+6 (+A). 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), go to location BUD+5 (+8); 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-4 (-8/-7); 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
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	$DFSM
0700	Execute overlay	$EXCOV
0701	Load overlay	$LDOV
0703	Status overlay	$STOV
070C	Unload overlay	$UNOV
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	

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

Monitor Call Number	Function Description	Macro Call Name
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
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

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

Monitor Call Number	Function Description	Macro Call Name
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
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
1402	Account identifier	$ACTID
1404	System identification	$SYSID
1B00	Set dial	$SDL

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

Monitor Call Number	Function Description	Macro Call Name
---	Clock request block template - generate	$CRB
---	Clock request block template - create	$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 - generate	$IORB
---	Input/Output request block template - create	$IORBD
---	Parameter structure block - generate	$PRBLK
---	Request block template	$RBD
---	Return sequence - establish	$RETRN
---	Semaphore request block - generate	$SRB
---	Semaphore request block template - create	$SRBD
---	File information block - offsets	$TFIB
---	Task request block - generate	$TRB
---	Template task request block - create	$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 MANUAL

ORDER NO. CB21, REV. 0

DATED NOVEMBER 1977

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

FOLD ALONG LINE

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

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

Honeywell

Honeywell Information Systems
In the U.S.A.: 200 Smith Street, MS 486, Waltham, Massachusetts 02154
In Canada: 2025 Sheppard Avenue East, Willowdale, Ontario M2J 1W5
In Mexico: Avenida Nuevo Leon 250, Mexico 11, D.F.

20677, 3478, Printed in U.S.A.

CB21, Rev. 0