Honeywell

PLANNING AND BUILDING AN ONLINE APPLICATION

SERIES 60 (LEVEL 6)

GCOS/BES2

SOFTWARE

Honeywell

PLANNING AND BUILDING AN ONLINE APPLICATION

SERIES 60 (LEVEL 6)

GCOS/BES2

SUBJECT:

Considerations for Planning and Building an Online Application Using Series 60 (Level 6) GCOS/Basic Executive System 2 Software.

SOFTWARE SUPPORTED:

This publication supports Release 0200 of the GCOS/Basic Executive System (BES2). When a later release of the system occurs, see the Subject Directory of the latest Series 60 (Level 6) GCOS/BES2 Software Overview and System Conventions manual (Order No. AU50) to ascertain whether this revision of this manual supports that release.

DATE:

July 1976

ORDER NUMBER:

AU49, Rev. 0

PREFACE

This manual describes some of the considerations involved in planning and building an online application using GCOS/BES2 software. Unless stated otherwise, the term BES refers to GCOS/BES2 software; the term Level 6 indicates the specific models of Series 60 (Level 6) hardware on which the described software executes. GCOS/BES2 software executes on the 6/30 Models of Series 60 (Level 6).

Section 1 presents the general characteristics of the various categories of BES software, and their roles in the development of an online application.

Section 2 provides details of the capabilities of BES software, presents formulas for calculating the sizes of various data structures, and introduces such concepts as priority levels and logical resource numbers that provide both flexibility and efficiency in the completed application.

Section 3 describes the use of the Configuration Load Manager. It includes a summary chart of the building process from the beginning stage where file space is allocated, through program development, to configuration and execution of the online application.

Section 4 provides practical advice on debugging an online application.

Appendix A contains complete descriptions of the CLM commands and their operands.

Appendix B discusses the use of Executive object modules in building an online application.

Appendix C presents examples of application configuration.

 File No.: 1S13

GCOS/BES2 SUBJECT DIRECTORY

This subject directory is designed to assist the user in finding information about specific topics related to GCOS/BES2. Topics are listed alphabetically; each topic is accompanied by the order number of each manual in which the topic is described. At the end of the Subject Directory, all GCOS/BES2 manuals are listed according to the alphabetic/numeric sequence of their order numbers.

Subject	Order No.
Allocate Disk File (Utility Set 1)	AU47
Application Development (Overview)	AU50
ASCII Character Set and Conversion Tables	AU50
Assembling Programs	AU48
Assembler Diagnostic Flags	AU43
Assembly Source Language	AU43
BASIC	AU44
Bootstrap Generator	AU47
Bootstrapping and Loading	AU46
Buffer Manager	AU45
Building an Online Application	AU49
Card Loader	AU46
Clock Manager	AU45
COBOL Compilation	AU48
COBOL Source Language	AU42
COBOL Statements	AU42
COBOL Compiler Diagnostic Messages	AU42
COBOL Operating Procedures	AU46
Command Processor	AU48
Communications	AU45
Compare Disk Volumes/Files/Members (Utility Set 3)	AU47
Configuration Load Manager	AU49
Console Messages (Error and Informational)	AU46
Control Panel	AU46
Copy Disk Volume/File/Member (Utility Set 3)	AU47
Cross-Reference Program	AU48
Debugging (Offline)	AU47
Debugging (Online)	AU49
Delete Disk File/Member (Utility Set 1)	AU47

Subject	Order No.
Disk Conventions	AU50
Disk Loader	AU46
Dumps	AU47
Dump Edit	AU47
Editor	AU48
Equipment Requirements	AU50
Error Reporting by Online Applications	AU46
Error Reporting by System Software	AU46
Executive Components	AU45
File Manager	AU45
File Naming Conventions	AU50
Floating-Point Simulator	AU45
FORTRAN Compilation	AU48
FORTRAN Compiler Diagnostic Messages	AS32
FORTRAN Intrinsic Functions	AS32
FORTRAN Source Language	AS32
FORTRAN Statements and Procedures	AS32
Glossary of System Terms	AU50
Hexadecimal Numbering System	AU43
Initialize Disk Volume/File (Utility Set 1)	AU47
Input/Output Drivers	AU45
Linker	AU48
List Disk Volume/File Description (Utility Set 1)	AU47
Loaders	AU46
Macro Facility Usage	AU43
Macro Preprocessor	AU48
Offline Applications	AU45
Operating Procedures	AU46
Operator Interface Manager	AU45
Overlay Loader	AU45
Paper Tape Loader	AU46
Planning an Online Application	AU49
Print Disk File/Member (Utility Set 2)	AU47
Program Development Tools	AU48
Program Naming Conventions	AU50
Program Patch	AU47
Punch Disk File/Member to Paper Tape (Utility Set 2)	AU47
Rename Disk Volume/File/Member (Utility Set 1)	AU47
Replace Memory Values (Utility Set 1)	AU47
Scientific Branch Simulator	AU45
Software Release Materials (Contents)	AU50
System Conventions	AU50

Subject	Order No.
System Software and Documentation (Overview)	AU50
Task Manager	AU45
Trace Trap Handler	AU45
Transfer Input to Disk File/Member (Utility Set 2)	AU47
Trap Handling (Offline)	AU46
Trap Handling (Online)	AU45
Utility Programs	AU47

The following publications constitute the GCOS/BES2 manual set. The subject Directory in the latest Series 60 (Level 6) GCOS/BES2 Software Overview and System Conventions manual lists the current revision number and addenda (if any) for each manual in the set.

Order No.	Manual Title
AS32	Series 60 (Level 6) GCOS/BES FORTRAN Reference Manual
AU41	Series 60 (Level 6) GCOS/BES2 COBOL Reference Manual
AU43	Series 60 (Level 6) GCOS/BES2 Assembly Language Reference Manual
AU44	Series 60 (Level 6) GCOS/BES2 BASIC Reference Manual
AU45	Series 60 (Level 6) GCOS/BES2 Executive and Input/Output
AU46	Series 60 (Level 6) GCOS/BES2 Operator's Guide
AU47	Series 60 (Level 6) GCOS/BES2 Utility Programs
AU48	Series 60 (Level 6) GCOS/BES2 Program Development Tools
AU49	Series 60 (Level 6) GCOS/BES2 Planning and Building an Online Application
AU50	Series 60 (Level 6) GCOS/BES2 Software Overview and System Conventions

In addition to the GCOS/BES2 manual set, the following manual is required by GCOS/BES users as a general hardware reference:

Order No.	Manual Title
AS22	Honeywell Level 6 Minicomputer Handbook

The following manual provides detailed information regarding programming for the Multiline Communications Processor:

Order No.	Manual Title
AT97	Series 60 (Level 6) MLCP Programmer's Reference Manual

CONTENTS

		Page
Section 1	Introduction	1-1
Section 2	Planning	2-1
	Overview of BES Software Services	2-1
	Services Available for Application Execution	2-1
	Services Available for Application Development	2-1
	Defining Application Design Objectives	2-3
	Defining Online Environment Characteristics .	2-4
	Selecting System Variables	2-4
	Information for System Data Structures From CLM Commands	2-4
	Size Calculations for System Data Structures	2-6
	Selecting Executive Modules	2-8
	Selecting Input/Output Modules	2-8
	Selecting File and Buffer Management Techniques	2-8
	File Manager Buffer Handling	2-10
	Buffered Read Operations	2-10
	Buffered Write Operations	2-11
	Interactive File Type/LFN Coordination ..	2-12
	Printer Space Conventions	2-12
	Designing Programs for an Online Environment.	2-13
	Multitasking	2-13
	Priority Levels	2-13
	Logical Resource Numbers	2-15
	Attaching LRN's to Levels	2-16
	Requesting Tasks	2-17
	Input and Output Drivers	2-18
	Memory Usage Considerations	2-19
	Hardware Dedicated Locations	2-19
	Data Structure Areas	2-19
	Overlay Planning	2-19
	Establishing Overlay Areas	2-21
	Overlay Coding Conventions	2-21
	Example of Nonfloatable Overlays	2-22
	Example of Floatable Overlays	2-24
	How to Estimate Overlay File Size	2-25
	Initialization Subroutines	2-25
	Communications Planning	2-27
	Priority Level Requirements for Communications	2-27
	Requesting Communications Functions	2-28
	Binary Synchronous Communications (BSC 2780)	2-29
	IBM 2780 Remote Terminal Emulation	2-29
	Level 6-to-Level 6 File Transmission ..	2-29

		Page
Section 3	Building	3-1
	Preparing to Use CLM	3-1
	Output File Preallocation (Stage 1)	3-3
	Source Module Creation and Editing (Stages 2 and 3)	3-3
	Object Module Creation (Stage 4)	3-4
	Load Module Creation (Stage 5)	3-4
	Linking Order for Code Text	3-4
	Externally Defined Symbols	3-4
	Summary of Load Module Preparation	3-6
	Using the Configuration Load Manager (Stage 6)	3-6
	How to Include Optional CLM Extensions	3-6
	Application Configuration and Loading	3-8
	Nonstop Application Loading	3-8
	Loading From Disk Using the Command Processor	3-9
	Load and Halt Procedures for Disk	3-9
	Loading From Disk With an Operator's Console	3-9
	Loading From Disk Without an Operator's Console	3-10
	Loading From Paper Tape	3-10
	Building a CLM Command File	3-12
	CLM Action During Loading	3-13
	Starting an Online Application	3-17
	Assembly Language Start Address Definition.	3-17
	FORTRAN Language Start Address Definition .	3-17
	COBOL Language Start Address Definition ...	3-18
Section 4	Debugging	4-1
	Using the Online Debug Program	4-1
	Online Debug Program Functions	4-2
	Debugging Command Language	4-2
	Debugging Command Format and Symbology ..	4-3
	Debugging Commands	4-5
	Activate Level Command (AL)	4-5
	All Registers Command (AR)	4-5
	Assign Command (AS)	4-5
	Clear Command (C*)	4-5
	Clear Command (Cn)	4-6
	Change Memory Command (CH)	4-6
	Define Command (Dn)	4-6
	Display Memory Command (DH)	4-7
	Dump Memory Command (DP)	4-7
	Define Trace Command (DT)	4-7
	Execute Command (En)	4-8
	GO Command (GO)	4-8
	Print Header Line Command (Hn)	4-8
	List All Breakpoints Command (L*)	4-9
	Line Length Command (LL)	4-9
	List Breakpoint Command (Ln)	4-10
	Print Command (P*)	4-10
	Print Command (Pn)	4-10
	Print Trace Command (PT)	4-10
	Reset File Command (RF)	4-10
	Set Breakpoint Command (Sn)	4-11
	Specify File Command (SF)	4-11

		Page
Section 4 (cont)	Set Level Command (SL)	4-12
	Set Temporary Level Command (TL)	4-12
	Print Hexadecimal Value Command (VH)	4-12
	Using the Online Debugging Program	4-13
	Additional Operating Notes for the Online Debug Program	4-14
	Locating Load Modules	4-15
	Debugging During Online Application Development	4-16
	Monitor Points	4-16
	Manual Control	4-17
	Real-Time Clock (RTC)	4-17
	Data Structures	4-18
	Trace History	4-19
	Handling Load Errors	4-20
Appendix A	Configuration Load Manager Commands	A-1
	Command Format	A-2
	Input Devices for CLM	A-3
	ADMOD Command (Add Load Module)............	A-4
	ATFILE Command (Attach File)	A-4
	ATLRN Command (Attach LRN)	A-5
	BSC 2780 Command	A-6
	BUFSPACE Command (Pool Definitions)	A-7
	CLOCK Command (System Clock)	A-8
	COMM (Communications System Command)	A-9
	DATE Command (Date and Time)	A-9
	DEVFILE Command (File Management Devices) .	A-10
	DEVICE Command (I/O Device Task)	A-11
	ELACT Command (End Load Action)	A-13
	ELOC Command (Define Address Symbol)	A-14
	EQLRN Command (Equate LRN's)	A-14
	EVAL Command (Define Value Symbol)	A-14
	FILMGR Command (File Manager)	A-15
	FMDISK Command (File Management Disk)	A-15
	IOS Command (I/O Stream)	A-15
	LACT Command (Load Action)	A-16
	LTPDEF Command (LTP Definition)	A-16
	LTPn Command	A-17
	MODEM Definition Command	A-18
	OIM Command (Operator Interface Manager Definition)	A-19
	QUIT Command (Initiate Loading)	A-19
	STATION Command	A-20
	SYS Command (System)	A-20
	TASK Command (Define Task)	A-21
	TRAP Command (Trap Vector)	A-21
	TSA Command (Trap Save Area Definition) ...	A-22
	TTY Command	A-22
	VIP Command	A-23
	*Command (Comments)	A-24
Appendix B	Planning and Building With Executive Object Modules	B-1
	Creating Executive Load Modules	B-2

		Page
Appendix C	Application Configuration Example	C-1
	Configuration Commands for Sample Input/Output Application	C-1
	Link Commands for Sample Input/Output Program	C-1
	Sample Input/Output Program	C-1
	Configuration Commands for Sample Communications Application	C-7
	Link Commands for Sample Communications Program	C-7
	Sample Communications Program	C-7

ILLUSTRATIONS

Figure 2-1.	Sample LRN Priority Level Attachments	2-15
Figure 2-2.	Sample Statements Attaching LRN's to Levels	2-16
Figure 2-3.	Memory Data Structures	2-20
Figure 3-1.	Building an Online Application - Process Diagram.	3-2
Figure 3-2.	Current Load Module Memory Layout	3-5
Figure 3-3.	Memory Layout During Configuration	3-14
Figure 3-4.	Memory Layout After Loading	3-15
Figure 3-5.	Memory Layout During Application Execution	3-16
Figure 4-1.	Sample ZXMAP Output	4-16
Figure 4-2.	Hardware/Executive Data Structures	4-18
Figure B-1.	Initialization Processing	B-2
Figure B-2.	New Initialization Modules	B-3

TABLES

Table 2-1.	BES Software for Application Execution	2-2
Table 2-2.	BES Software for Application Development	2-3
Table 2-3.	Physical and Logical Resource Requirements	2-3
Table 2-4.	Effects of CLM Parameters on Memory Usage	2-5
Table 2-5.	Names and Sizes of Honeywell-Supplied Load Modules	2-9
Table 2-6.	Relative Priority Level Assignments	2-14
Table 2-7.	Register Use by System Initialization Subroutines	2-26
Table 3-1.	CLM Functional Groups, Component Modules and Related Commands	3-7
Table 3-2.	Bootstrap Record for Nonstop CLM Loading	3-8
Table 3-3.	CLM Load Module Order for Paper Tape	3-11
Table 4-1.	Memory and Work File Space Usage	4-1
Table 4-2.	Summary of Debugging Commands by Function	4-2
Table 4-3.	Symbols Used in Debugging Command Lines	4-4
Table A-1.	Summary of CLM Commands and Command Functions	A-1
Table B-1.	Executive Object Modules	B-1

SECTION 1

INTRODUCTION

This manual shows you how to combine BES software modules with your own programs to achieve the functionality you require in your online application.

The software has a number of features that enable you to concentrate on the solutions to your specific application problems, rather than to spend time developing, coding, and testing your own standard service routines.

For example, Executive modules provide routines for task and clock management, and for the control of operator dialog with the software. Also included are routines for time and date recording, as well as trap and error handling. There is a set of routines that allows you to execute your application as a series of overlays – the Overlay Loader provides the basic capabilities for loading and starting overlay code.

The input/output modules provide file management facilities, and reentrant routines for driving all available devices. Moreover, the software includes Communications modules that function through the standard physical I/O interface, and can be configured and used as easily as any other peripheral device.

Furthermore, the modularity of the software allows you to choose only those services that you require. For example, you may or may not want to include the Executive buffer management capability as an integral part of your configured application.

Since the software is supplied in load module form, you do not have to assemble and link the various routines that you want to include in your application. As soon as you have finished developing your programs so that they are executable load modules, you are ready to configure your application.

The process chart in Section 3 gives you an overview of the operations involved in building your application. You will be using various functional groups of software modules in the building process. For example, in setting up your application environment, you will use the utility programs to prepare and maintain disk volumes. See the Utility Programs manual for details.

You will use components described in the Program Development Tools manual to prepare your own application programs.

To make a realistic debugging environment for online application programs, the software provides an Online Debug Program that operates under the Executive.

When your own load modules are ready for use, you configure your online application by using a software component called the Configuration Load Manager (CLM). The CLM, working in conjunction with a loader, provides a load-and-go capability for your application. For smaller systems, CLM can be executed as a series of overlays.

CLM accepts command input in which you have specified the various characteristics of your application, such as memory size, numbers and types of devices, and the load modules, both BES modules and your own, to be included in the final configuration. CLM uses this command information to build the data structures and to load the necessary modules that the Executive software uses to control processing. A special facility permits user-written application initialization during loading.

After all the specified modules are loaded, CLM turns control over to the highest priority task that has been designated as initially active, and application execution begins.

SECTION 2
PLANNING

Planning an online application includes the process of systems analysis and design, taking the fullest advantage of the system services and development tools supplied by Honeywell. This process includes:

- Acquiring a familiarity with the capabilities of BES software
- Defining application design objectives
- Defining online environment characteristics
- Designing programs to be run in an online environment

OVERVIEW OF BES SOFTWARE SERVICES

BES provides a number of services that you should consider when planning an online application. There are also a variety of tools for use in the development of application programs. These software modules are summarized in Tables 2-1 and 2-2 below.

Services Available for Application Execution

The BES-provided services for use during online application execution are summarized in Table 2-1, and described in detail in the Executive and Input/Output manual or the FORTRAN manual, as appropriate.

Services Available for Application Development

The BES-provided tools for use during application development are summarized in Table 2-2, and described in detail in the Program Development Tools and Utility Programs manuals, as appropriate.

In addition to the services summarized in Table 2-1, BES provides a Configuration Load Manager (CLM) which defines the context of the application, sets up the required internal data structures, loads the specified modules, and initiates execution of the application, all in one continuous operation. The operation of the CLM and the syntax of its commands are described in this manual.

Table 2-1. BES Software for Application Execution

Service	Description
Task Manager	Monitors and controls all tasks in the application, using data structures defined during configuration. The Task Manager oversees the level activity indicators, administers the interrupt structure, and coordinates requests for the execution of tasks.
Clock Manager	Activates priority levels after an elapsed time interval or at regular time intervals, as specified during configuration. An expanded Clock Manager is available, providing date and time information in ASCII format for external reporting.
Operator Interface Manager	Controls all operator dialog with the software through a KSR-like device.
Buffer Manager	Coordinates requests for buffer space, using control structures and buffer pools defined during configuration. Use of the Buffer Manager is optional but recommended.
File Manager	Opens, reads, positions, writes, and closes files; reports file status information and error conditions. Use of the File Manager is required for FORTRAN and COBOL programs.
Overlay Loader	Controls the loading of planned overlays through data structures created by CLM.
Communications Supervisor	Provides necessary services to communications handlers including time-out, request validation, and Task Manager interface.
Device Drivers (Disk, Printer, Card Reader, KSR/ASR, VIP, BSC, TTY, MLCP)	Perform all data transfers between the online application and its respective devices. Drivers receive requests for service from the Task Manager and run at the priority level of the requested device.
Floating-Point Simulator	Provides double-precision arithmetic and a scientific instruction set. Operates as a trap handler under the Executive.
Scientific Branch Simulator	Operates as a trap handler under the Executive to provide FORTRAN and assembly language programs with the means to simulate the use of scientific branch instructions.
Trace Trap Handler	Traces the contents of a specified memory location and maintains a limited trace history.
FORTRAN Run-Time I/O Routines (FRIOR)	Provides for reading and writing of formatted and unformatted records; edits integer, real, logical, and character data for formatted input and output, and produces diagnostic messages for inappropriate commands. These routines require the use of the File Manager.
Online Debug Program	Provides online program testing and patching facilities for application programs running under a BES Executive.
COBOL Run-Time Routines	Supports the COBOL procedural statements that involve arithmetic, logical, data manipulation, and input/output operations.

Table 2-2. BES Software for Application Development

Component	Function
Utility Programs	File maintenance and handling, media transfers, printing, debugging.
Editor	Creates and/or corrects source programs on disk.
Macro Preprocessor	Provides for definition and expansion of macro routines. Macro Preprocessor output is input to the Assembler.
Assembler	Produces object modules from source programs written in BES assembly language.
COBOL Compiler	Produces object programs from source programs written in COBOL.
FORTRAN Compiler	Produces object modules from source programs written in FORTRAN.
Linker	Produces load modules from object text output of all language processors.

DEFINING APPLICATION DESIGN OBJECTIVES

The careful description of the specific design objectives of your application is an important part of the overall planning process. You should have a precise inventory of the numbers and kinds of problems that your application programs will be designed to solve, the files required, and the types of reports to be generated.

This inventory will greatly simplify your assessment of the physical and logical resources required to achieve your design objectives. Your inventory should contain information about the source language in which your particular application program is written because high-level languages such as COBOL and FORTRAN require the File Manager to handle the logical input/output operations between programs and the physical devices that contain the data read and written by those programs. Table 2-3 shows one way of relating a design objective to the physical and logical resources needed to implement that objective.

Table 2-3. Physical and Logical Resource Requirements

Design Objective	Application Program Source Language	Resources Needed	
		Logical	Physical
Produce a report based on daily card input	COBOL	Input task Processing task Output task File Manager	Card Reader Disk device Line printer

There are other considerations about the use of logical and physical resources that are discussed throughout the remainder of this section. For example, the implications of providing certain values in the CLM commands, and the use of overlays to conserve memory space. There is a description under "Designing Programs for an Online Environment," later in this section, about the use of logical resource numbers to coordinate the use of interrupt priority levels by tasks and devices.

DEFINING ONLINE ENVIRONMENT CHARACTERISTICS

The characteristics of the online environment such as priority level usage, trap handling routines, time and date functions, as well as the complement of Executive software and file and buffer specifications, are chosen by supplying information to the CLM in configuration commands. These commands are described in detail in Appendix A; some of their more salient features are presented in this section.

Selecting System Variables

The information provided in the various commands to the CLM affects the numbers and sizes of control structures used by the Executive software to monitor processing. Choice of Executive services (and hence the Executive modules) has a direct bearing on the amount of available memory remaining for the loading of application programs. Table 2-4 summarizes the relationships between parameter values of the CLM commands and effects of these values on the sizes of control structures and memory usage. (See also "Size Calculations for System Data Structures," below.)

Information for System Data Structures From CLM Commands

Note that the CLM control commands direct the action of the CLM itself; the load configuration commands provide information to the loader that pertains to the loading order of the modules, and the identification of references among them.

The system configuration, task, buffer management, and file management commands contribute information for the control structures that are used to regulate processing in an online application, in addition to providing CLM with data needed to calculate the sizes of these control structures.

Table 2-4. Effects of CLM Parameters on Memory Usage

Parameter	CLM Command	Range of Values	Default Value	Default Value (Words)	Structures Affected
hilrn		up to 255	15	16	LRT size
lolevel	SYS	6≤value≤62	15	270	Number of ISA's
himem		up to 64K	Loader Address		-
lrn	OIM	≤hilrn	-	-	LRT and ISA
level		5≤value≤lolevel	-		
number of TSA's	TSA	2≤value≤46	2		Number of trap save areas
size		8≤value	8	16	Size of trap save areas
trap number	TRAP	1 through 46	-		See Table 2-5 for sizes
handler name		ASCII name			
module name	ADMOD	-	-		See Table 2-5 for sizes
maxlfn		≤255	15	32	IORB
				224	FDB
concurrent calls				56	FCB
	FILMGR	>0	4	272	REB
concurrent opens				328	Diskette buffer
		>0	8	392	Cartridge disk buffer
size, number	BUFSPACE	-	-		Size of PPT
lrn	TASK, ATLRN, DEVICE, TTY, VIP, BSC	≤hilrn	-		Number and size of RCT's
level		5≤value≤lolevel			

The effect of some of the parameters in Table 2-4 on the amount of memory space used is small, so that if a default value is taken even when it is higher than that needed for a particular parameter, not much memory space is wasted. However, you should be careful about assigning higher than necessary values to the parameters for the file management commands, particularly the FILMGR command. As you can see by inspecting the size calculation formulas that use information from this command, a careless assignment of values to the "concurrent calls" and "concurrent opens" parameters will result in the reservation of much more space than needed.

Moreover, the lolevel parameter value should be selected with care. The interrupt save areas (ISA) are set aside on the basis of this parameter, so that if you supply a value of 30, and actually only use 10 priority levels, 360 words remain unused.

SIZE CALCULATIONS FOR SYSTEM DATA STRUCTURES

The system and task commands provide information for the calculation or the definition of the sizes of the following data structures:

LRT - Logical resource table
ISA - Interrupt save areas
SOQ - Start of queue header table
EOQ - End of queue header table
TSA - Trap save area
RCT - Resource control table

The following formulas are used to calculate the sizes of these areas.

$$S_{LRT} = (\text{hilrn} + 1)$$

If the default value is taken for hilrn, the size of this table would be 16 words.

$$S_{ISA} = MIN + (\text{lolevel} + 1-4) * MAX$$

Where MIN = 6 and MAX = 22. If the default value of 15 is taken for lolevel, the ISA would be 270 words long.

$$S_{SOQ} = S_{EOQ} = (\text{lolevel} + 1)$$

Using the default value for lolevel, each of these tables would be 16 words long.

$$S_{TSA} = (\text{number of blocks}) * (\text{blocksize})$$

Using the default values for these parameters results in a trap save area 16 words long.

A resource control table for a device is 16 words long; an RCT for a task is one word long.

In summary, the size of the area devoted to those data structures defined by the system and task commands is:

$$S = S_{LRT} + S_{ISA} + S_{SOQ} + S_{EOQ} + S_{TSA} + N_D * S_{DEV} + S_{RCT}$$

where N_D is the number of DEVICE commands, and S_{RCT} is the sum of RCT sizes.

The buffer and file management commands provide information for the definition of the following data structures:

- PPT (pool parameter table)
- Buffer area
- Work area for File Manager

- IORB (input/output request block)
- Diskette buffers
- FDB (file descriptor block)
- FCB (file control block)
- Remote extent block
- Device buffers
- Wait table
- Semaphore table
- VDB (volume descriptor block)
- LFT (logical file table)
- FCB (device)
- FDB (device)

The following formulas are used to calculate the sizes of these structures.

S_{PPT} = 4 words + (2*P)

where P is the number of size/number pairs indicated in the BUFSPACE command.

$S_{buffer\ area}$	= $(size_1 * number_1) + \ldots (size_n * number_n)$
$S_{work\ area}$	= (number of concurrent calls) * 40
S_{IORB}	= (number concurrent calls) * 8
$S_{diskette\ buffer}$	= (number concurrent calls) * 72 + 64 for cartridge disk
S_{FDB}	= (number concurrent opens) * 28
S_{FCB}	= (number concurrent opens) * 7
S_{REB}	= (2 * number concurrent opens) * 17
$S_{device\ buffer}$	= (number double buffers) * 77
$S_{wait\ table}$	= lolevel + 1
S_{ST}	= 16 + (2 * (number VDB + number FDB_{device})) + maxlfn + 2 + (2 * number concurrent opens)
S_{VDB}	= (number FMDISK commands) * 48
S_{LFT}	= 1 + maximum lfn + 1
$S_{FCB\ (device)}$	= (number ATFILE commands) * $(6+1+N/_2)$

where N is the pathname length, space-filled to the next highest even number of bytes.

$S_{FDB\ (device)}$ = (number DEVFILE commands) * 28

Selecting Executive Modules

BES software provides an Executive (ZXEX03) as a load module. This Executive supplies capabilities for task and clock management, input/output handling, overlay loading, initialization processing, operator interaction with the software, time and date recording, trace trap and system error handling. Applications that use the File Manager require this Executive.

To complement the facilities provided by the Executive, you can also include either the File Manager or the Buffer Manager, or both of these load modules in your application. All FORTRAN applications require the File Manager.

You may find that the load module version of the Executive does not exactly fit your application specifications. In that case, you can hand tailor an Executive load module from Honeywell-supplied object modules. See Appendix B for a description of this process.

Selecting Input/Output Modules

Input and output operations in your online application are handled by software modules called device drivers. You may select appropriate device drivers (or line-type processors for Communications devices) from the set provided by Honeywell (in either load or object module form), or you may write your own device drivers. See the Executive and Input/Output manual for details of this process.

Table 2-5 contains the names and sizes of the Honeywell-supplied load modules.

Selecting File and Buffer Management Techniques

If your online application is programmed in assembly language, application requirements determine whether the File Manager is required (both FORTRAN and COBOL programs require the File Manager) for input and output operations.

When your application needs only minimal I/O functions, you can gain a space advantage by using the device driver alone and save the 3200 words occupied by the File Manager. However, you then must program whatever I/O functions your application does need. The advantage of using the File Manager is that it provides these needed functions, and hence, I/O processing with the programming simplicity of a higher level language.

The following discussion illustrates the advantage of using the File Manager over a device driver alone when the I/O device is a disk.

Table 2-5. Names and Sizes of Honeywell-Supplied Load Modules

Load Module Name	Description	Approximate Size in Words (Decimal)
ZXEX03	Executive	2600
ZXBM01	Buffer Manager	100
ZYFM02	File Manager	3200
ZIDSK	Diskette Driver	175
ZICDSK	Cartridge Disk Driver	225
ZIKSR	Keyboard-Send-Receive Driver	450
ZIASR	Automatic-Send-Receive Driver	1200
ZICDR	Card Reader Driver	125
ZILPT	Printer Driver	125
ZXOVLY	Overlay Loader	275
ZDBG	Online Debug Program (overlay version)	1200
ZQEXEC	Communications Supervisor	580
ZQMLON	MLCP Driver	500
ZQPTTY	TTY Line-Type Processor	980
ZQPVIP	VIP 7700 Line-Type Processor	1270
ZQPBSC	BSC 2780 Line-Type Processor	1200
ZFPSIM	Floating-Point Simulator	400
ZFBSIM	Scientific Branch Simulator	250
TRPHND	Trace Trap Handler	150
NOTE: The names of the object module versions are listed in Appendix B.		

When using a device driver alone to perform input/output operations, the application program must build its own data structures to interface with the driver, and initialize those structures with the data that the driver needs in order to locate the required data on the device. This information consists of the initial sector to be transferred given by the sector number relative to the beginning of the volume, and the number of bytes to be transferred.

After the I/O request is made, the driver will transfer the requested number of bytes starting at the boundary of the sector specified. The implication of dealing with sectors it twofold: the application program must know, for each set of data (logical record) the number of the sector in which the record resides. Secondly, if there is more than one logical record per sector, the program must do its own deblocking; i.e., must find the logical record it needs.

By contrast, the File Manager builds the I/O data structure for the program and provides the initializing information by which the logical record is retrieved when provided with a record number relative to the beginning of the file.

On a read operation, the File Manager transfers the requested data record, not the physical sector, into the program's buffer area, and the program then does not have to search for and deblock the records itself.

To summarize, the File Manager works at the program's logical level with files and records; a driver works at the hardware physical level with sectors. When using nondisk sequential devices, the File Manager provides some measure of device independence at the application interface.

FILE MANAGER BUFFER HANDLING

The File Manager allows nondisk devices to be buffered. When you configure your application, you can request the File Manager to reserve internal space for a data buffer for a particular LFN (see the DEVFILE command in Appendix A). The purpose of this buffering is to allow application code to execute in parallel with I/O transfers.

File Manager achieves parallel processing in two different ways, depending on whether the LFN is used for obtaining data from a device (reading), or transferring data to a device (writing).[1]

Buffered Read Operations

A buffered read results in an anticipatory read in addition to every read command issued by the application. When the LFN represents a card reader, this means that a second card will be read immediately after the application reads (and waits for the data to arrive in its buffer) the first card after the LFN is opened. The application can now process the data it has while the physical I/O transfer for the next card into the File Manager's buffer is in progress.

Nondisk file types recognized by the File Manager may be classified as interactive or noninteractive. Interactive device names are: KSR, KSI, KSO, ASR, ASI, ASO, VIP, VIPI, VIPO, TTY, TTYI, TTYO (see DEVFILE command in Appendix A).

Buffered interactive and noninteractive file types operate exactly the same <u>after</u> they are opened. A noninteractive file, when opened, does not initiate an anticipatory read by the File Manager. This means that the application must wait for the physical I/O transfer to occur on the <u>first</u> read; thereafter the parallel operation described above, occurs.

[1]Bidirectional LFN's cannot be buffered.

An OPEN command to an interactive file type does cause an anticipatory read into the File Manager's buffer to occur. If the application program immediately followed the OPEN with a READ command, the effect is exactly the same as for a noninteractive file type; i.e., the application is suspended until the read operation into the File Manager's buffer completes and data is moved into the application program's buffer. However, the application program can avoid suspension on the first or subsequent READ commands by using the Status Read command, which gives the program immediate information as to whether the File Manager's buffer now contains the next record. If not, the application program can continue processing and only perform the read operation when the result of the Status Read indicates that data is available.

The primary use for buffering an interactive file type is to allow an application to control input from more than one LFN, each of which represents a console at which operators enter data. The application program cannot perform a READ on any particular LFN, and wait until data arrives because the operator at that terminal may not be present, and the application program is then indefinitely suspended. To avoid this indefinite suspension, the Status Read should be used, in this way the application program will never perform a READ unless data is present. (It is because of the indefinite response on an input LFN that the OPEN command to an interactive file type causes the anticipatory read, thus the Status Read is meaningful for all read operations, not only for the first one.)

Buffered Write Operations

A buffered write operation to an LFN works on behalf of the application program in the same logical manner as the read – the program is permitted to execute in parallel with the physical I/O transfer to the device. To achieve this parallel processing, no special operation occurs on an OPEN command, and no distinction is made between interactive and noninteractive file types. Each write command is completed by moving data from the application buffer to the internal File Manager's buffer, initiating the transfer, and returning control to the application program. If the program performs a second write operation while the internal buffer is still in use for a previous transfer, the application is suspended until the buffer is available and new data moved into it again. The application can avoid suspension by using the Status Write command to see if the internal buffer is still in use or not.

Special considerations for buffered write operations arise because, if a physical I/O error occurs while data is being transferred from the internal buffer to the device, the application program is unaware that an error has occurred unless it checks the file status after each write. Furthermore, if an error does occur, the application program may need to have saved (or be able to retrieve) the data record so that it can be repeated.

To summarize, a Status Write should be used to test for buffer availability and no error status before each write operation (not required on the first write operation) or the close operation of the file.

INTERACTIVE FILE TYPE/LFN COORDINATION

Using the File Manager to provide application programs (including those written in FORTRAN or COBOL) with read and write capabilities for KSR-like devices requires two LFN's, used as a pair. Both LFN's represent the same physical device, one for the keyboard input, and the other for the printer output. Two LFN's are required because the read LFN must be buffered if more than one terminal is being controlled, and a bidirectional, buffered file type is not supported by the File Manager.

Whenever both LFN's are buffered or not depends on the application's needs. However, when the terminal being controlled is physically attached to a communications controller (MLCP), great care must be taken to coordinate the closing of the files involved. If one is closed before the other is finished using the terminal, the other LFN will be unable to access the device because the close operation causes the connection to be broken. (A media error is returned to the application.) For further information, see the discussion of CONNECT (used by the File Manager OPEN function), and DISCONNECT (used by the File Manager CLOSE function) in the Executive and Input/Output manual.

PRINTER SPACE CONVENTIONS

In planning an application that uses line printers and terminals (consoles) interchangeably, you must consider the differences in format conventions between these two types of devices.

The line printer driver (and the IORB within the File Manager for the LPT device-file) assume that a space-before-print convention is appropriate. The device-specific word and the format control byte allow convenient prespacing.

The teleprinter driver allows prespacing but also supports post-line feed operations usually associated with console-oriented print-then-space conventions. This convention is designed to allow input to begin on a new line without doing a line feed after a key is struck.

The File Manager's preconfigured IORB associated with any KSR-like device assumes a post-line feed is desirable.

The application can accommodate either convention, by itself, without difficulty, but if the devices are interchangeable, care must be taken to avoid either:

- Double spacing because the format byte specifies prespace one line, and the teleprinter IORB enforces a post-line feed

- Overprinting because the format byte specifies no prespace, and the line printer does not support post-line feed

The distinction as to device type is made, using the File Manager, by interrogation of file type status. Physical I/O can discover the difference by locating the RCT for the particular LRN and testing the device ID.

DESIGNING PROGRAMS FOR AN ONLINE ENVIRONMENT

As you design your application programs, remember that they will be using some of the same system resources that are used by the Executive software. For this reason, your application programs should conform to certain conventions that make the joint usage of these resources as efficient and error-free as possible. These conventions concern the use of interrupt priority levels, the definition of control structures, the use and saving of registers, as well as the standard ways of defining, identifying, and calling the various Executive and application modules.

Multitasking

The following paragraphs describe what has to be done to set up your application for multitasking execution. Appendix C contains a sample program that illustrates configuration, linking, task control blocks, and tasking.

A task is a sequence of executable code whose execution is initiated and terminated by calling task management functions described in the Executive manual. When several tasks can be active simultaneously you have multitasking. The criterion used by firmware to select a task for execution, from among those have been initiated and are active, is the task's priority level; the task associated with the highest priority level is the one to be scheduled next.

PRIORITY LEVELS

Each task and device (i.e., the device driver task) is associated with a priority level number, reflecting its relative processing priority in an application. In this priority scheme, the lower the level number the higher the priority. Table 2-6 contains a list of possible relative priorities for tasks. Level 0 through 4 (the five highest priority levels) and level 63 (the lowest level) are reserved for system use and do not need to be specified during configuration. All other levels are available for use by application program tasks and devices.

Table 2-6 includes all the devices that currently are supported on Level 6 hardware. If fewer devices are used, fewer levels are needed while maintaining the relative position of the levels. It is suggested that consecutive levels be used, without skipping a level number, to save data structure space that would otherwise be reserved for the unused levels.

Table 2-6. Relative Priority Level Assignments

Level	Use
0	Power failure handler
1	Watchdog timer runout
2	Trap save area overflow
3	Inhibit interrupts
4	System clock
	Communications interrupt
	Communications Devices (less than or equal to 9600 bps)
	Cartridge disks
	Communications Devices (less than or equal to 1200 bps)
	Diskettes
	Printers
	Card readers
	ASR/KSR
	Online Debug Program
	Operator Interface Manager interrupt
	Input/output - bound application tasks
	Central processor - bound application tasks
63	System idle loop (always active)

The table indicates I/O devices, and not device drivers, to stress that each (noncommunications) peripheral device must have at least one level assigned to it; peripherals cannot share a level. If there are two printers, each must be assigned a unique level. Actually, when a device level is initiated, it is a reentrant I/O driver that is initiated of which only one copy need be in memory.

Communications requires one nonshareable level dedicated to processing communications interrupts, and it must be at a higher level than any communications devices. Communications devices can share a level. For example, four TTY's and one VIP can either share one level or be configured to use up to four levels.

The listed priority arrangement is designed to provide maximum throughput for each device by assigning the high transfer rate devices a higher priority than the lower transfer rate devices. I/O-bound tasks are run at a higher priority than central processor-bound tasks since this enables I/O-bound tasks, which run in short bursts, to issue I/O data transfer orders as needed, wait for I/O completion, and while in the wait state, relinquish control of the central processor to the central processor-bound tasks. Otherwise, if the central processor-bound tasks had a higher priority, the I/O devices would be idle while I/O-bound tasks wait to receive central processor time. The criteria used to specify Table 2-6 might not suit a particular application and the level assignments should be modified to include other priority considerations.

LOGICAL RESOURCE NUMBERS

To enable an application program to be independent of level numbers, the software provides logical resource numbers (LRN's) to associate application tasks and devices with priority levels.

An LRN (and not level number) is given with each task request to indicate the level at which the task is to execute. The level at which the task executes is determined finally by the level that was attached to the LRN at configuration time. If level changes are to be made, the application only has to be reconfigured with the new level; the program does not have to be changed.

Although an LRN is attached to a unique level, more than one LRN can be attached to the same level when LRN's are used synonymously (e.g., two independently created tasks refer to the same task by different LRN's), or when tasks or communications devices share the same level.

Figure 2-1 illustrates an association between tasks of an application and priority levels. The first column describes the task to be initiated. During task initiation the task is associated with one of the LRN's in the second column which was attached, at configuration time, to one of the priority levels listed in the third column. The level assignments follow the priority scheme listed in Table 2-6.

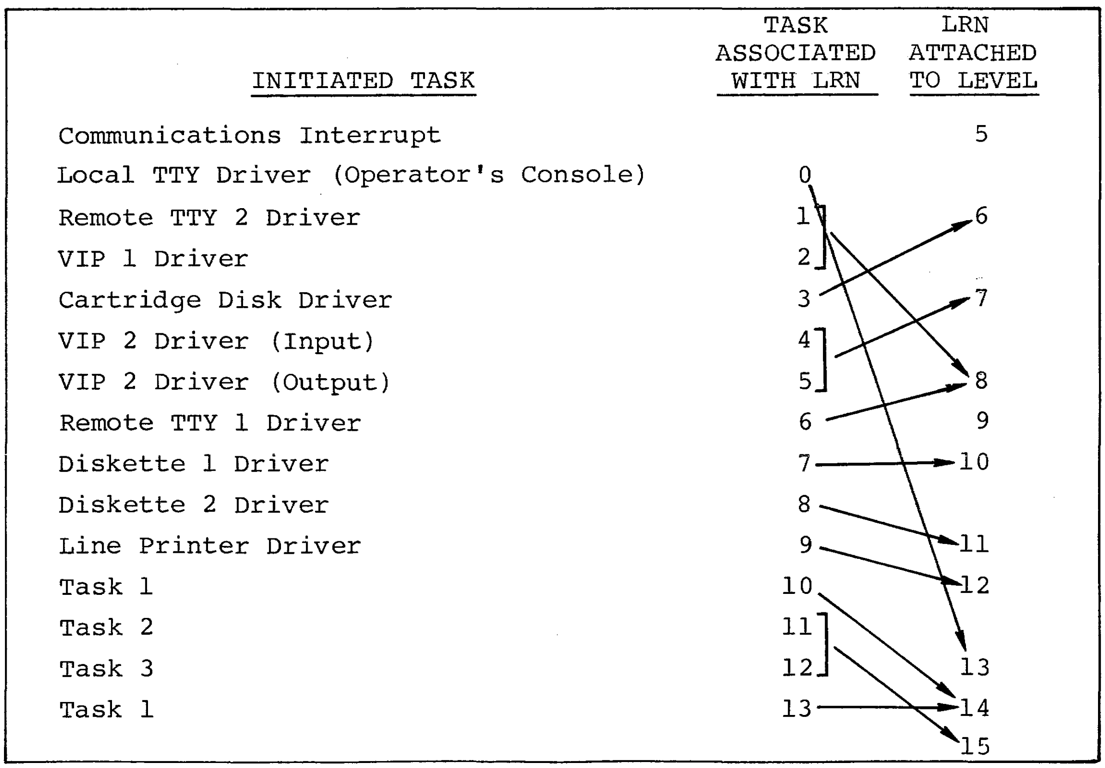


Figure 2-1. Sample LRN Priority Level Attachments

Starting with the tasks at the top of the figure, the operator's console by convention uses LRN 0. Following it are two communications device drivers, each with a separate LRN and sharing one level with LRN 6. VIP 2 has different LRN's for input and output, but it must have the same level. This allows a different configuration to use different devices without changing the program. Further down, two diskettes have unique level assignments, since peripheral devices cannot share a level. Level 9 is unused. Application tasks can share a level, and tasks 2 and 3 share level 15. Task 1 can be initiated by using LRN 10 or LRN 13, both referring to level 14. Although it might appear otherwise, the order of the task-LRN association is almost arbitrary. Levels 0 through 4 are dedicated assignments that do not require CLM statements. Level 5 though not attached to a LRN must be specified using the CLM COMM command.

ATTACHING LRN's TO LEVELS

The CLM ATLRN command is used to attach one LRN to one level. The DEVICE command does the same for peripheral devices. Each communications device has a unique command. An example of the commands used to specify the relationship illustrated in Figure 2-1 is given in Figure 2-2.

```
                                        .
                                        .
                                        .
OIM      0,13
COMM     5                        Communication Interrupt Level 5
DEVICE   KSR,0,13,X'0500'         Operator's Console Level 13
TTY      1,8,X'FD80'              Remote TTY Level 8
VIP      2,8,X'FC00'              Remote VIP Level 8
DEVICE   FCD,3,6,X'1280'          Cartridge disk (fixed) Level 6
DEVICE   RCD,14,6,X'1280',3       Cartridge disk (removable) Level 6
VIP      4,7,X'FC80'              Remote VIP input Level 7
EQLRN    5,7                      Remote VIP output Level 7
TTY      6,8,X'FD00'              Remote TTY Level 8
DEVICE   DSK,7,10,X'1300'         Diskette Level 10
DEVICE   DSK,8,11,X'1380'         Diskette Level 11
DEVICE   LPT,9,12,X'0580'         Line printer Level 12
ATLRN    10,14                    Task 1 Level 14
ATLRN    11,15                    Task 2 Level 15
EQLRN    12,15                    Task 3 Level 15
ATLRN    13,14                    Task 4 Level 14
                                        .
                                        .
                                        .
```

Figure 2-2. Sample Statements Attaching LRN's to Levels

The cartridge disk used in the example has a fixed and a removable disk, and needs two DEVICE commands. The parameter "3" is required to cross reference the previous disk DEVICE command LRN.

The EQLRN command is used when a new LRN has the same level as another LRN. For example:

```
ATLRN  11,15
EQLRN  12,15
```

The ATLRN with an RCT-size parameter enables you to specify another RCT for a level. This feature could be used to create an RCT for a nonstandard device. The program would then have to initialize the RCT with data that the CLM normally enters from the CLM Command parameters; e.g., channel number, modem.

None of the statements in Figure 2-2 will cause execution to start after loading. To start execution immediately after loading, the TASK command must be used with the fourth parameter set to YACT. If TEST01 (a task) with LRN 10 were to be activated after loading, the TASK command would be:

```
TASK TEST01,10,14,YACT
```

REQUESTING TASKS

To have task A request task B for initiation, task A calls Task Management's "request" routine, and passes it the address of task control block (tcb) containing task B's start address and LRN. The tcb is built and initialized by the calling task, task A. If more than one task is to be initiated at the same priority level, the first task requested is the first one to be executed, without interruption from other tasks at the same level. The LRN used in the call to the "request" routine can be attached to a level during configuration either by using the ATLRN or TASK commands. However, if a task is to be executed immediately after loading, it must be defined in a TASK command.

An alternate means for requesting a task can be used when the task is to have exclusive use of a level. Instead of obtaining a start address from the tcb, Task Management uses the B5-register contents of the interrupt save area (ISA) of the priority level of the requested task. A bit set in the tcb by the calling task indicates to the Task Manager whether to use a start address in the tcb or ISA. To initially set the B5-register to a desired task start address, the TASK command must be used. CLM takes the start address given in the command and places it in the B5-register of the ISA.

If a task, after it terminates, is to be called again using the address in the ISA, the terminating task must contain a terminating code sequence that permits the B5-register in the ISA to be restored to the desired start address. Such a code sequence of a terminating task is given below.

```
A     LDV $R1,=130
A+1   LNJ $B5,<ZXTERM
A+2   (first instruction of task being terminated)
```

The context of the level is saved in the ISA whenever a task terminates at a level. In the above terminating code sequence, the B5-register contains the address A+2, which is the desired start address of the task.

Input and Output Drivers

The input/output operations in your application are handled by software components called drivers (for conventional peripheral devices) or line-type processors (for communications devices). These components perform the following general functions:

- Initiate I/O operations on individual devices
- Report errors and status information
- Monitor timing to detect device failure or inactivity
- Perform limited editing of transferred information

See the Executive and Input/Output manual for descriptions of the drivers and the control structures they use.

Honeywell supplies device drivers in both load and object module form. The names and sizes of the driver load modules are given in Table 2-4. The procedure for linking object module device drivers is described in Appendix B of this manual.

A Honeywell-supplied driver is implicitly loaded when CLM processes a DEVICE command. The DEVICE command provides the logical resource number and the priority level for the specified device. Similarly, the line-type processors are implicitly invoked when the appropriate communications device command (TTY, VIP, BSC) is processed.

If you write your own device driver, implicit invocation does not occur, and in order to include the module in CLM's load list, you must include an explicit ADMOD command in the configuration command file that builds your application. Furthermore, if your device driver requires nonstandard commands and parameters, you must provide the interpretive routines that build the control structures required by your driver as extensions to the CLM.[1]

The Honeywell-supplied drivers and line-type processors are reentrant, so that only one copy of the driver appears in the final configuration.

---

[1]Consult the current Release Bulletin for details about CLM extensions.

Memory Usage Considerations

The memory area used by an online application consists of hardware-dedicated locations, data structure areas, load module residence areas, buffer and loader areas, and areas occupied by the symbol table and the CLM on a transitional basis. The total size of these areas determines the memory size requirements of an application.

NOTE: In the descriptions that follow, all memory locations are specified in hexadecimal notation, unless otherwise indicated.

HARDWARE DEDICATED LOCATIONS

Low memory, from location 0 through location 00BF, is reserved for BES use. Among the indicators and pointers stored in this area are the trap save area pointer (word 0010), clock information (words 0014, 0015, and 0016), the level activity indicators (words 0020 through 0023), the trap vectors (words 0052 through 007F), and the interrupt vectors (words 0080 through 00BF). A detailed memory layout and explanation of contents of the hardware-dedicated locations appears in the Executive and Input/Output manual.

DATA STRUCTURE AREAS

Immediately above the hardware-dedicated locations is the data structure area. During the configuration process, the CLM builds the data structures required by the online application in this area of memory. Using the information supplied in its commands, the CLM determines the sizes of tables and save areas, constructs the framework of various tables, and inserts into those tables the information that is available at the time. The data structure area begins at location 00C0 and extends as far as necessary to accommodate the required structures.

Figure 2-3 illustrates the layout and contents of memory.

OVERLAY PLANNING

The overlay technique allows you to economize on memory by using a given portion of it over and over again; it also forces you to think critically about the nature of the solutions to your application problems, and the order in which those solutions are achieved.

The Overlay Loader consists of a set of reentrant service routines that provide the basic capabilities required for loading and starting overlay code. (See the Executive and Input/Output manual for details.)

The Overlay Loader resides in memory during application execution; it controls overlay processing by using an overlay file and a set of data structures that were created by CLM as a result of information in the bound unit (root and overlay segments) produced by the Linker.

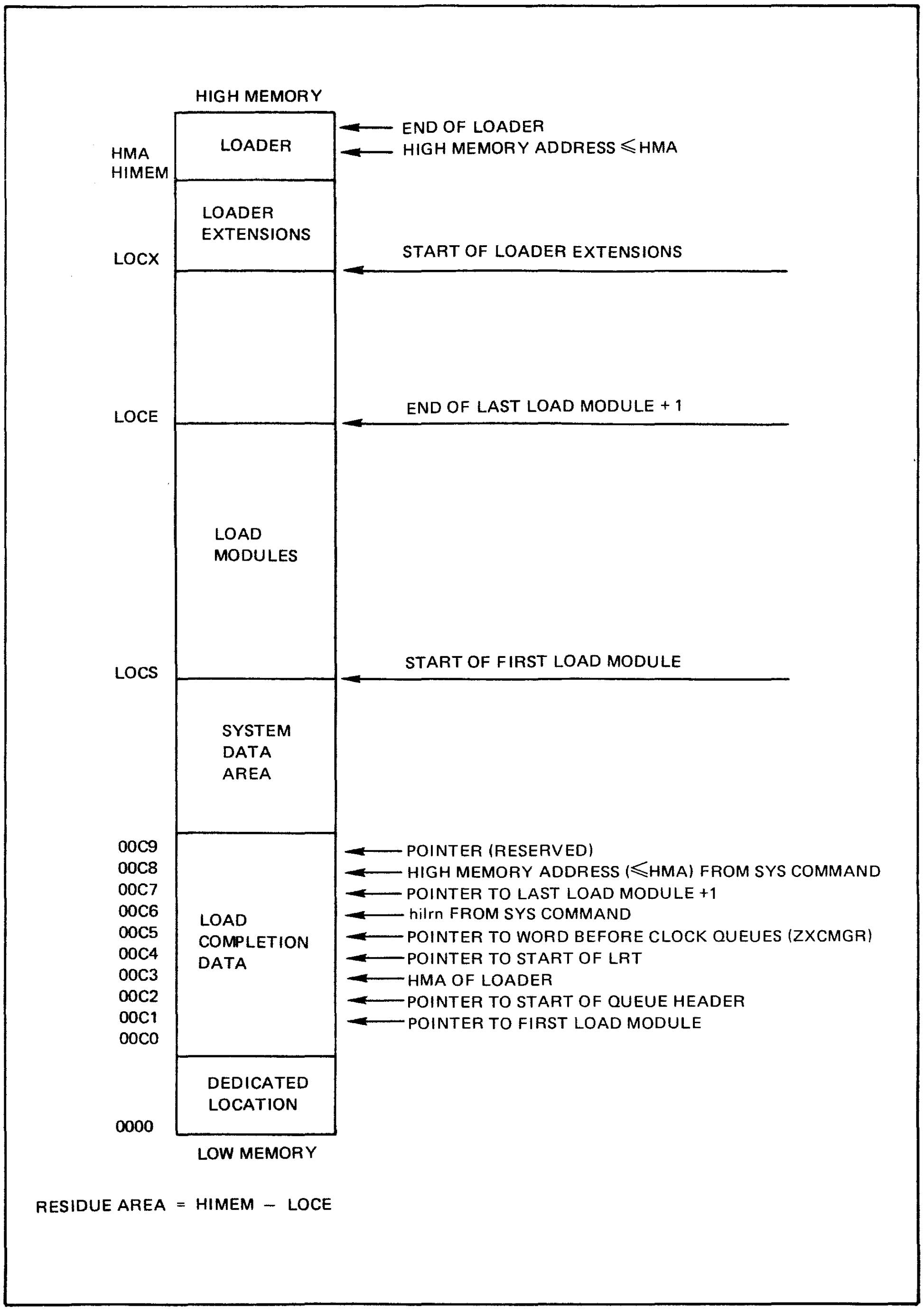


Figure 2-3. Memory Data Structures

Establishing Overlay Areas

The items and locations used in the following discussion are shown in Figure 2-3.

Theoretically, all of memory above the system data area (LOCS) should be available for use by programs that execute as overlays. There are some limitations, however.

Apart from the fact that the root module of a bound unit must be resident during execution, thus limiting the actual area that can be used by the overlays, there is a property of overlay code produced by the higher level language compilers, and even some types of code written in assembly language that makes unrestricted use of all available memory impossible. Such code is called "nonfloatable." (Refer to Program Development Tools manual for details.)

Normally, CLM treats all overlays as if they were nonfloatable; that is, it loads them into memory in exactly the area from which they will eventually be executed. The total area required for a set of nonfloatable overlays is the area required by the largest nonfloatable overlay module. The first example below shows a bound unit that has two nonfloatable overlays.

If the overlay code is floatable, that is, dynamically relocatable when it is reloaded for execution, the overlay does not contribute to the overall load space, and it is possible to use the area above the end of the last load module (LOCE) in which to position the floatable overlays. This is particularly important when LOCE and LOCX are nearly the same value; then floatable overlays can be executed in the area reserved by CLM during loading.

Perhaps the simplest way to set aside space for floatable overlays is to incorporate the Buffer Manager in your configuration and request blocks of memory equal to the size of overlay code in the BUFSPACE command to CLM.

Alternatively, the CLM residue above LOCE can be used by developing specific addresses in the root after all loading is complete based on information in the CLM-created "Load Completion Data Area" in Figure 2-3. The pointers to LOCE and the high memory address of the loader are useful for developing load addresses to position floatable overlays.

Overlay Coding Conventions

The use of overlay processing requires an understanding of the way in which root and overlay segments are defined for processing by the Linker, and the relationships between root and overlay segments.

Modules to be processed as overlays are identified as such when they are linked. The following Linker commands identify the root modules and overlays, and specify the position of overlays in relation to the root module.

NAME - Identifies the root module.

IST - Marks the location of initialization code in the root.

OVLY - Names the overlay module.

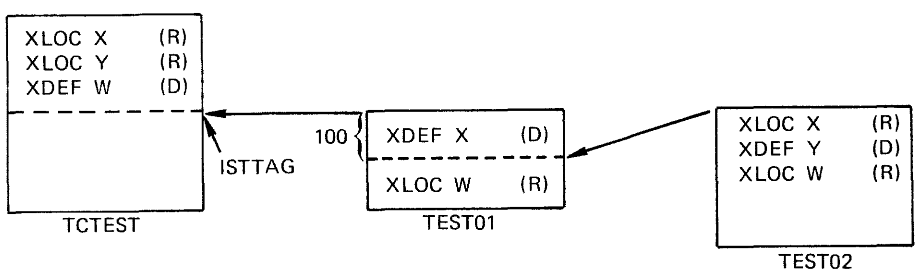
BASE - Positions the overlay module.

In response to information placed in the load module by the Linker when it processes these commands, the CLM constructs a relative file containing the overlay modules; the root module remains memory resident. CLM also builds the data structures that the Overlay Loader uses to manipulate overlays during the execution of the application. Refer to the Linker portion of the Program Development Tools manual for details about creation of bound units and symbol definitions.

Example of Nonfloatable Overlays

This example shows a root program, TCTEST, that has some initialization code labeled ISTTAG, and two overlays: TCTEST01, and TCTEST02. The diagram below shows the relationship between the root program and the two overlays; the letters in the modules indicate symbols that are either defined (D) in a module, or referred to (R).

When it is loaded, overlay TCTEST01 is located at ISTTAG; overlay TCTEST02 is located at ISTTAG+100, as shown below.



The Linker commands to create the bound unit are:

NAME	TCTEST	Name used in the ADMOD command.
LINKN	TCTEST	Links root.
EDEF	W	Defines symbol externally referred to from outside the bound unit (not shown).
IST	ISTTAG	Defines initialization code and overlay position.
OVLY	TCTEST01	Names first overlay; Linker writes root to disk.
BASE	ISTTAG	Indicates position of first overlay.
LINKN	TEST01	Links first overlay.

EDEF X	Defines externally referenced symbol.
OVLY TCTEST02	Names second overlay; Linker writes first overlay to disk.
BASE ISTTAG+100	Indicates position of second overlay.
LINKN TEST02	Links second overlay.
EDEF Y	Defines externally referenced symbol.
END	Completes definition of bound unit; Linker writes second overlay to disk.

Notice that only one IST command is used — only root segments may use initialization code. The Linker can satisfy the reference from the first overlay to W in the root because the root is linked first. However, a reference to Y from TCTEST01 would cause a CLM error halt — the Linker writes TCTEST01 to disk with Y as an unresolved symbol, and CLM will not write an overlay to its file if it contains an undefined symbol.

A reference from an overlay to W in the root is legitimate because the Linker retains all symbol definitions not purged by subsequent BASE commands affecting the same area. References from the root to X and Y defined in the overlays require EDEF statements in the overlay command group because the CLM must resolve the references when these modules are loaded. (The Linker was unable to resolve the references before the root load module was written to disk.)

The EDEF definition for W in the root module is superfluous for this example, but illustrates another requirement for symbol definition, namely that an EDEF statement is needed when a symbol is referred to from outside its own bound unit.

When the application using the bound unit shown in the example above is configured, the CLM receives this ADMOD command:

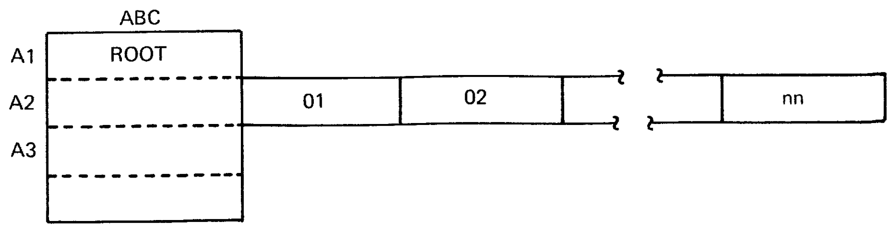
ADMOD filename:TCTEST ...

As a result, the CLM will load the root segment and its initialization code into memory following any code loaded as a result of previous ADMOD statements. The initialization code beginning at ISTTAG is executed <u>before</u> loading the first overlay. Then, using the information specified in the BASE command for the first overlay, that segment is loaded. If the overlay has no undefined symbols, it will be written out to a temporary file.

Finally, the second overlay is loaded starting at ISTTAG+100, and written out to the temporary file. The CLM continues to process command statements.

Example of Floatable Overlays

This example illustrates a bound unit whose root has dedicated areas within it, and whose overlay segments are all floatable (dynamically relocatable).



The Linker commands to create this bound unit are:

NAME ABC	Provides name to be used in the ADMOD command.
LINKN A	Links root segment.
OVLY ABC01	Names the first overlay; Linker writes root to disk.
BASE A1	Locates overlay.
LINKN ABC01	Links first overlay.
OVLY ABC02	Names second overlay; Linker writes first overlay to disk.
BASE A1	Locates overlay.
LINKN ABC02	Links second overlay.
⋮	
OVLY ABCnn	
BASE A1	
LINKN ABCnn	
END	

The overlays in this example are all floatable, so that the code in the root and the overlays could include load addresses developed from the CLM-supplied pointers in low memory as described earlier.

Note that the ADMOD command for ABC must be positioned in the CLM command sequence in such a way that the largest floatable overlay to be written out by CLM may be loaded into memory below location LOCX (Figure 2-3). This can be accomplished by having other ADMOD commands, whose modules occupy at least as much space as the largest ABC overlay, follow the ADMOD for ABC.

After loading is completed, the overlays are brought into memory areas previously occupied by CLM by calling the Overlay Loader.

The CLM determines the final size of a bound unit (and thereby the location where the next load module begins) based on the highest address occupied by either the root or one of its nonfloatable overlays. This means that the root alone in the floatable overlay example determines the bound unit size. If the root does not include overlay areas within it (e.g., by using a RESV assembly statement), those areas must be obtained in alternate ways.

Care must be taken in source code to produce a floatable overlay. However, an overlay may inadvertently be coded (or later modified) so that it matches the definition of a floatable overlay. To prevent a change in CLM operation resulting from a nonfloatable overlay becoming floatable, the source code of the nonfloatable overlay could include a global reference to an address tag within that same source.

How to Estimate Overlay File Size

The CLM assumes that there is enough physically contiguous space in the relative file it uses for the application overlays. If this is not true, portions of the disk beyond the allocated file space will be destroyed. The CLM also assumes that the name of the relative file it will use to contain the overlays is either OVERLAY, or the filename used in the AT03 command to the Command Processor.

To estimate how many sectors should be allocated when using the utility initialize function, take these steps:

1. Produce link maps for all overlay load members.
2. Determine the number of words in the image text of each one.
3. Divide the image text word total by the number of words per sector (64 for diskette; 128 for cartridge disk) to obtain the number of sectors for each overlay.
4. Add the individual sector requirements together to get the total.
5. Add in the Online Debug Program sector requirement (22 for diskette; 22 for cartridge disk), if needed. (This is always a good idea, because you may need the ODP later.)

It is important to note that each overlay begins on a sector boundary so that it may be read into memory (and written out by the CLM) as a single I/O transfer. This design minimizes overlay load time during execution.

INITIALIZATION SUBROUTINES

Initialization subroutines may or may not be required by every module. As you write the initialization code for your modules, you may want to use one or more of the system-provided subroutines summarized in Table 2-7. These subroutines use the standard register conventions that you will also use when you write your subroutines.

Table 2-7. Register Use by System Initialization Subroutines

Module Name	Function Code	Function	Register Contents: For Function Call	Register Contents: After Function End
ZGFINU	0	Find a symbol in symbol list	B4 - Pointer to start of symbol name	R1 - 0 = Symbol found nonzero = Not found
ZGDEFU	1	Define a symbol	R1 - 0 = Address definition Value = Value definition R2 - Definition value B3 - Definition address	R1 - 0 = No error 1F = Symbol already defined 21 = Work area overlap
ZGREFU	2	Refer to a symbol	R1 - 0 = Address reference Value = Value reference B1 - Pointer to location referred to	R1 - 0 = No error 21 = Work area overlap

NOTES: 1. These registers have the same values for all functions: R3 - function code; B4 - pointer to the start of the symbol name; B5 - return address.

2. Other registers used by these subroutines: R4, R7, B2, B7, R6.

Initialization is performed immediately after the loading of a module is completed. Each initialization subroutine is entered via the LNJ instruction using register B5 for the return address. The address of the parameter list is loaded into register B4, the parameter list itself is defined in the initialization subroutine table described below.

Errors occurring during initialization are fatal errors; loading cannot be continued. Error information is returned in register R1; CLM moves the contents of R1 to R2 and places the 1304 halt code into R1. If the content of register R1 is zero, the operation was successful.

The initialization subroutine table identifies the subroutines that are to be executed when the module has been loaded. It has the following format:

```
label    DC      0        next load displacement
         RESV    $AF,0    RFU
         DC      <name    first initialization subroutine
         DC      value    parameter 1} for first subroutine
         DC      value    parameter 2}
         DC      <name    second initialization subroutine
         DC      value    parameter 1} for second subroutine
         DC      value    parameter 2}
          .        .                  .
          .        .                  .
          .        .                  .
         RESV    $AF,0    sentinel, end of IST
          .        .                  .
          .        .                  .
          .        .                  .
```

Entries must be included in the initialization subroutine table for each subroutine required for a load module. The "label" in the first statement of the format example must be defined in an IST command to the Linker. The location at "label" is the point at which the next module will be loaded when the initialization is completed. Normally, the value declared at "label" is zero.

During the CLM loading phase, the base address for loading the next load module is formed by using the address of the first word of the current load module's initialization subroutine table (IST in the example) plus the displacement value contained in that word. When the displacement is nonzero, the next module loads below (for a negative displacement) or above (for a positive displacement) the IST start address.

Communications Planning

The Communications functions of BES software have been designed in such a way as to make them as easy to use as any other peripheral device. Your interaction with the Communications software occurs through the physical I/O interface. Using the Configuration Load Manager, you assign a logical resource number to the various Communications devices. Then, in your application program, using a standard call to the Executive, and providing a standard control structure (an IORB) in which to pass parameters, you request a transaction with a particular communications resource. Your request is then handled by the Communications software, and you need not be concerned with the details of Communications procedure. See the IORB information in the Executive and Input/Output manual for details about the standard control structures and function codes for Communications devices.

PRIORITY LEVEL REQUIREMENTS FOR COMMUNICATIONS

Although both peripheral and Communications devices share a common interface, they have different priority level requirements. Peripheral devices such as card readers, disks, and printers are assigned one device to a level. Communications devices, however, require one dedicated level (specified in the COMM command) that is reserved for the processing of Communications interrupts, and must be the highest priority level assigned to a Communications function. Additionally, any number of priority levels may be shared among Communications devices (not with any other device types); these priority levels must be lower (higher level numbers) than the level specified in the COMM command.

REQUESTING COMMUNICATIONS FUNCTIONS

When you request a transaction with a communications resource, you must specify the logical function in the request block that you provide with each request.

There are five logical functions: connect, read, write, wait-on-line, and disconnect. The connect must precede other requests, because Communications resources are configured in a disconnected state. The sequence that would occur is as follows:

1. Set up an IORB with the function code for a connect request, and call the physical I/O interface.
2. Once the connection is made, you supply the appropriate request blocks for the functions that your application will perform, and do the reads, writes, and/or wait-on-line operations required by the program's logic.
3. When the program finishes processing, you supply a request block with the disconnect function code, and call the physical I/O interface to perform the function.

The values that you provide for the various function codes are coded in the last four bits of word three (ZIRCT2) of the IORB that you supply in your application program.

If your application is such that the program must:

- Temporarily suppress the previously queued data request to or from a VIP or TTY, or
- Signal a traffic direction change for a device (BSC)

there is a means of disconnecting the resource logically while maintaining the physical line connection. This logical disconnection is accomplished when bit 15 of the device-specific word of the IORB is set on; when this bit is zero, the physical line connection is discontinued.

The Communications function codes (CONNECT and DISCONNECT) may be used with no effect if a program whose IORB's contain such codes were to be executed using noncommunications peripheral devices, thus the program is independent of the device types that may be in use.

COBOL application programs can use the Communications facilities of BES software by the standard input/output verbs OPEN and CLOSE; these verbs evoke the Communications connect and disconnect functions, respectively.

BINARY SYNCHRONOUS COMMUNICATIONS (BSC 2780)

This Communications protocol may be used in conjunction with an appropriate application program in the following ways:

- IBM 2780 remote terminal emulator
- File transmission for Level 6-to-Level 6 computers

IBM 2780 Remote Terminal Emulation

In this environment an application program would be structured to emulate the functions of the IBM 2780 remote terminal in a manner which is consistent with the features available on the host computer responsible for processing the submitted data.

The features of BES2 BSC 2780 line protocol are described below.

Level 6-to-Level 6 File Transmission

In this environment an application program could be developed to transmit both binary and ASCII data, in records of any size, between two Level 6 computers.

The support of the character sets (whether ASCII or EBCDIC) is restricted to the line protocol handler providing the control characters in the appropriate character set.

The character set of the text portion of the data is totally the responsibility of the application program.

In transparent EBCDIC, the line protocol handler will assume total responsibility for inserting and removing the line protocol escape character (DLE) both in the header and in the text.

Whether a transmission unit from Level 6 will contain a single record or two records may be managed by the application program in the following way:

- Creation of single record transmission units requires that each write order be issued with the wait status specified in the IORB.
- Creation of two-record transmission units requires that each write order be issued with the do-not-wait status specified in the IORB.

  In this situation, the application would do the necessary processing and issue another write order with the do-not-wait status specified in the IORB. The process continues, the application program processes in a totally independent manner, without regard for the activity on the Communications line until the application program's write buffers are filled. At this point, a wait on the first IORB is issued. When the application program resumes, it again does its processing, and issues another write order (do-not-wait). Then a wait on the second IORB is issued, and so on.

The packaging of the write requests into a transmission unit is done independently by the line protocol handler. A second record will be embedded in the transmission unit as long as the write request is issued before the last character of the previous write request has been transmitted over the Communications line. As a practical matter, considering the comparative slowness of the communications line (maximum 1200 characters/second) with other resources (computer, peripherals, etc.), there is sufficient time to allow this free-wheeling process to work.

If an application program is by convention to be prepared to handle the receipt of data from an application that transmits two records in a single transmission unit, then it is required that two read requests always be present at any time during which a transmission unit may be received.

A basic characteristic of the BSC 2780 communications protocol is that it is nonconversational. That is, once the movement of data has been established between computers (i.e., <u>from</u> A to B), it is not possible to transmit from B to A until the entire quantity of data has been transmitted from A to B. Occasionally, there may be a need to send an urgent preemptive message in the direction contrary to the flow of information. This need is resolved by computer B issuing a disconnect request with the indicator set to abort all IORB's in the queue. Upon notification of this action being complete, it is now possible for the application program in computer B to issue a connect request followed by the write order for the urgent message.

When computer A received the notification from computer B of the urgent preemptive need to send a message (signaled in BSC 2780 by the receipt of a reverse interrupt, RVI) it would so notify the application program via the attention interface, after dequeuing and posting all currently queued IORB's. The application program would then issue a receive order for the acceptance of the preemptive message.

SECTION 3
BUILDING

The Configuration Load Manager (CLM) defines the application variables, sets up the required internal control structures, prepares a load list of the specified modules, and initiates the execution of the application, all in one continuous operation.

Before executing the CLM, you must have already prepared the programs and files to be used in the application. This preparation includes compiling (or assembling) the programs, linking them into one or more load modules, and preallocating space for all output files to be used.

During the configuration phase, CLM accepts the commands that direct its operation. In addition to specifying system characteristics such as memory size and processor type, the commands processed by CLM also set priority levels for tasks, assign logical resource numbers, and direct the construction of a load module list containing the names of all the modules to be loaded for execution.

Once the configuration phase is completed, the modules named in the load module list are loaded by the particular loader then in use. Any unresolved references among the modules are resolved at this time. As each module is loaded, it is initialized before the next module is loaded.

After the last module is loaded and initialized, control is transferred to the active task having the highest priority. Execution then begins.

PREPARING TO USE CLM

Since CLM allows an application to be run as a single load-and-go operation, all files to be used for output by the application should be preallocated, and all modules, both Executive as well as user-written modules, should be linked before the CLM is loaded into memory. These preliminary processes are described in the appropriate manuals, and illustrated in Figure 3-1. The process of building an online application falls naturally into discrete steps. The following pages describe the process and refer you to the pertinent manuals for complete details.

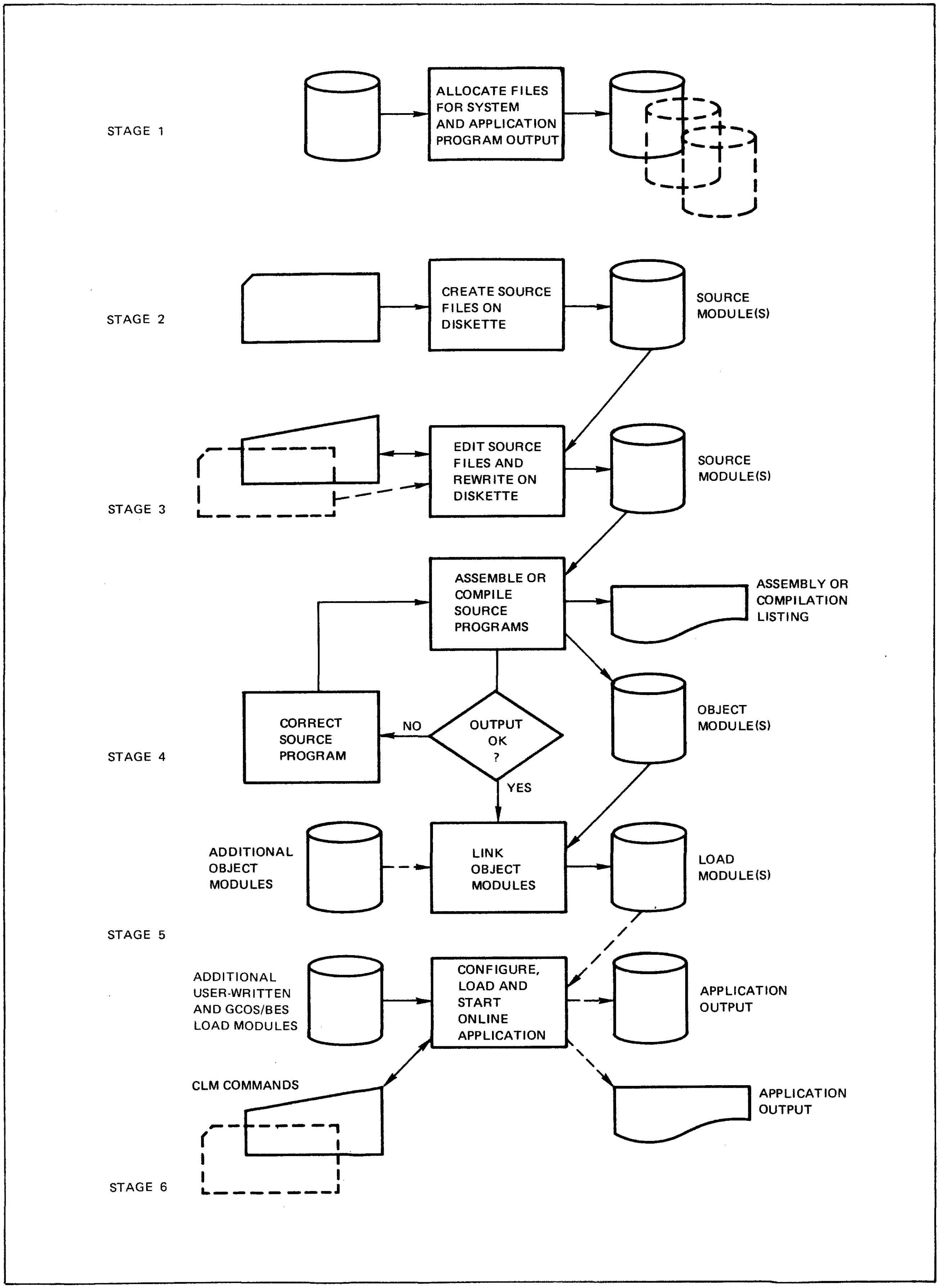


Figure 3-1. Building an Online Application - Process Diagram

Output File Preallocation (Stage 1)

Figure 3-1 summarizes the file requirements for all the following stages in the application development process. File space is allocated by Utility Set 1. (See the Utility Programs manual.) Some of the files your application uses must be relative files to receive either the output data from the application, or, if you are using overlays, the overlay file written by CLM.

The file to accommodate any overlays that CLM writes out in the process of application configuration must be a single extent, relative file large enough to hold all the application's overlays. If the overlay version of the Online Debug Program is being used, the overlay file must provide 50 diskette sectors, or 25 disk sectors in addition to the space required for other overlays.

Files that are intended to receive source, object, or load modules must be initialized after space is allocated, to organize those files as partitioned files capable of accommodating individually accessible members.

If you plan to use the Online Debug Program with predefined command lines stored on disk, you must preallocate a relative file named DEBUG.WORK containing 22 diskette sectors, or cartridge disk sectors for use by this program.

If your application program produces an output file that is intended for printing by a print utility, either immediately, or at a later time, the first byte of each record to be printed must contain printer control information.

Source Module Creation and Editing (Stages 2 and 3)

Partitioned files containing source text members are created on disk from punched card files using Utility Set 2. Once created, these source files are then usable by the Editor, for correction or addition of text, or they may serve as input to the Assembler or to the FORTRAN or COBOL Compiler. (See Program Development Tools manual.)

Stage 2 is mandatory if source programs are punched on cards; it may be omitted entirely if the source programs are short enough to be entered through the keyboard as input to the Editor.

Stage 3 is optional. It is possible to go directly from creating a source file on disk to the Assembler/Compiler phase.

Object Module Creation (Stage 4)

The creation of object modules is the function of three system programs: the Assembler for programs written in assembly language; the FORTRAN Compiler, and the COBOL Compiler, for programs written in FORTRAN or COBOL source language. In addition to object modules produced on disk, the Assembler and both compilers produce source listings with diagnostic messages that refer to the various syntactical errors encountered in the processing of the source language statements.

Once the source code is free of syntax errors, and has been reassembled or recompiled, the program is ready to be processed by the Linker in the next phase.

Load Module Creation (Stage 5)

The preparation of load modules for use in online applications requires special attention to the ordering of permanent code and the initialization code for particular load modules, and to the handling of externally defined symbols.

LINKING ORDER FOR CODE TEXT

One or more object modules may be linked to form a load module. The order in which modules are linked is significant in the following situations:

- Modules being linked require initialization code.
- Modules being linked will be executed as overlays.

Load modules that require initialization code must have all permanent code linked before the initialization code for the load module. This is because, during the loading phase in the operation of CLM, successive modules are loaded in such a way that once the initialization code for a module is executed, it is replaced by the permanent code of the next module. Figure 3-2 shows the memory layout of a module and its initialization routines, and indicates the starting location for loading the next module once the initialization has been performed.

EXTERNALLY DEFINED SYMBOLS

An application load module may have valid undefined symbols at the time it is linked, such as a call to an Executive subroutine. The CLM resolves these references as it loads the Executive Modules specified in the ADMOD, DEVICE, and Communications line type processor commands, and analyzes the ELOC, EVAL, TRAP and DEVICE commands.

Any symbol that may be referred to by other load modules or by the CLM, and not defined by CLM itself, must be identified in an EDEF statement at the time the module is linked. (See the Program Development Tools manual for Linker information.)

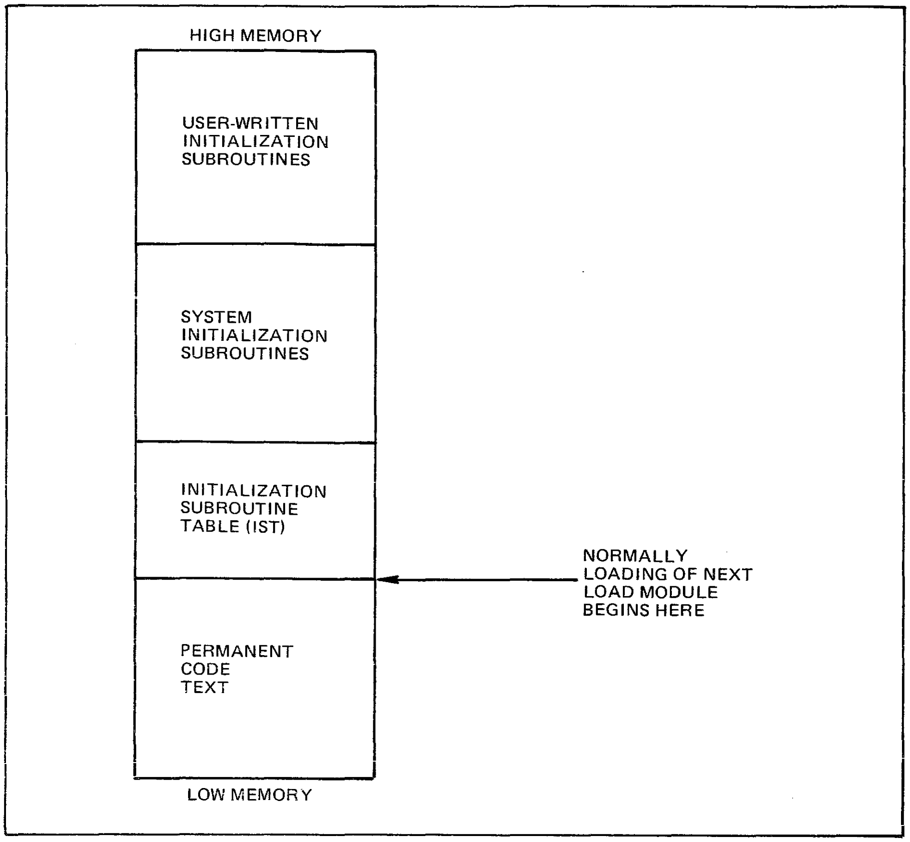


Figure 3-2. Current Load Module Memory Layout

In assembly language, locations or values that are referred to by modules other than the one in which they are defined, are identified in an XDEF statement; when the defining module is linked, the label(s) made available for external reference by these XDEF statements are declared in a Linker EDEF statement if CLM must resolve references to these labels in other modules.

During configuration, the CLM must be able to resolve all references either from information provided to it in a symbol table from the Linker, or from the symbol table CLM itself constructs from the command information submitted to it. The unresolved symbols encountered by CLM during execution cause a load error (1341 - see the Operator's Guide) followed by a halt; at your discretion, you can ignore these errors and continue processing. However, there are load errors that prevent continued execution of CLM: the occurrence of an undefined symbol in an overlay module (135B - see the Operator's Guide).

There are several CLM command parameters that require Linker EDEF statements: the start address in a TASK command; the ppt-label and space-name in the BUFSPACE command; the handler-name in a TRAP command; the label parameter of a DEVICE, TTY, VIP, or BSC command if the label specifies a user-defined routine and is not the default label, ZIATTN for LRN 0.

Summary of Load Module Preparation

These are the steps involved in the preparation of load modules prior to configuring an online application:

- Collect the object modules that make up the permanent code.
- Collect the system initialization modules to be used.
- Write and assemble the user-written initialization code and the initialization subroutine table.
- Run the Linker to produce a load module in the format described in Figure 3-2. Note that the initialization subroutine table is always linked immediately following the permanent code text. It is at this point in the process that the Linker EDEF command is used to specify all externally-referenced symbols.

USING THE CONFIGURATION LOAD MANAGER (STAGE 6)

The CLM and its extensions are the BES software components you use to configure an online application.

CLM consists of four functional groups of modules that interpret the commands in which you have specified the system variables, devices, and load modules that constitute the configured application. Of these functional groups, one, the CLM nucleus is required for configuring all applications; the other three are optional extensions that interpret particular configuration commands.

Depending on the memory size of your system, the optional modules may be resident throughout the operation of CLM, or, as with an 8K system, the CLM extensions must be executed as overlays. Table 3-1 summarizes the functional groups and indicates the commands that are interpreted by each.

How to Include Optional CLM Extensions

The CLM extensions that interpret the File and Buffer Management, and Communications commands are included by specifying the appropriate information in a LACT command for each extension. The LACT command contains a parameter to indicate that an extension is to be executed as an overlay. (See Appendix A.)

The following example shows the LACT commands for a communications application whose devices are accessed through the File Manager, and that will require device definitions that include the File Manager DEVFILE command.

```
LACT CLMCOMM:COMM
LACT PROGFILE:CLMFIL
```

The channel number is the same as that from which CLM was loaded, the work areas for both sets of modules will not be shared, and both sets of interpretive modules will be resident rather than overlays.

Table 3-1. CLM Functional Groups, Component Modules and Related Commands

Functional Group (filename:membername)[a]	Component Modules	Commands Processed
PROGFILE:CLM (required)	<u>CLM Nucleus</u> CLM CLM2 CLMST1 D$CLMST1 C$CLMST1 CLMST2 D$CLMST2	SYS ADMOD TRAP OIM PRMOD LACT TSA ELOC ELACT CLOCK EVAL QUIT DATE IOS * TASK EQLRN DEVICE ATLRN
PROGFILE:CLMFIL (optional)	<u>File Manager Extensions</u> CLMFIL D$CLMFIL C$CLMFIL	FILMGR DEVFILE ATFILE FMDISK
PROGFILE:CLMBUF (optional)	<u>Buffer Manager Extensions</u> CLMBUF D$CLMBUF C$CLMBUF	BUFSPACE
CLMCOMM:COMM (optional)	<u>Communications Extensions</u> COMM D$COMM C$COMM	COMM BSC TTY MODEM VIP LTPDEF LTPn STATION

[a]As specified in the LACT command

If the extensions are executed as overlays, it is not necessary, but more efficient to group the configuration commands in the same order as the extensions were specified. The grouping of commands becomes particularly important if your application is configured from a serial device. (See "Loading From Paper Tape," discussed later in this section.)

After the CLM nucleus has been loaded, the only commands that it will process are the LACT and IOS commands, until an ELACT command is read. In fact, even if you do not add any of the CLM extensions, an ELACT command must be issued so that CLM may begin processing the other application configuration commands.

To summarize; the optional CLM extensions may be executed as resident modules, or they may be executed as overlays; in an 8K environment the extensions <u>must</u> be executed as overlays; unlike the execution of <u>application program</u> overlays that require the availability of a disk, CLM extension overlays have no such requirement.

Application Configuration and Loading

You can load CLM and configure your application from disk or paper tape, either as a nonstop procedure (disk only), or a load-and-halt operation; you can use, but do not need, an operator's console. Specific operating procedures for all methods are described in the Operator's Guide, and discussed briefly in the following pages.

NONSTOP APPLICATION LOADING

The nonstop loading procedure is based on a preset bootstrap record created by the Bootstrap Generator utility program. The elements of this record are described in Table 3-2. (For details, see the Utility Programs manual.)

Table 3-2. Bootstrap Record for Nonstop CLM Loading

Parameter	Values	Default Values
DFT	N	None
BTHLT	N	N
HMA (high memory address)	1FFF (8K) 3FFF (16K) 7FFF (32K) FFFF (64K)	1FFF
KSR	0	0500
LDCHN (load channel)	0400 (disk) , (paper tape)	0
FILE	PROGFILE	PROGFILE
MEMBER	CLM	CMDPRC
REL (relocation factor)	XXXX[a] (8K) XXXX (16K) XXXX (32K) XXXX (64K)	0
LDHLT	N	N

[a]See Release Bulletin for exact values

The nonstop loading procedure is usable only when your load modules are on disk; no operator's console is needed.

The file requirements for nonstop loading from disk are these: the CLM command input file (CLMCI) must be on disk. If you are running your application program as overlays, you must preallocate a relative file for CLM to use when it writes out the overlays.

If you are configuring a communications application, then in addition to PROGFILE, your disk should contain CLMCOMM.

The Bootstrap Generator Utility program is executed to place the preset bootstrap record on the disk. The parameters for the utility, assuming a 16K system, are:

N,,3FFF,0,0400,,CLM,3480

When this record is on the disk, and all the load modules needed by the application are available, you are ready to carry out the loading procedure. You press: Stop, Clear, Load, and Execute; there will be a pause while the QLT (Quality Logic Test) is performed, then press Execute again, and application configuration is underway and needs no further intervention.

LOADING FROM DISK USING THE COMMAND PROCESSOR

This method involves minimal operator intervention, but allows the command input file to the CLM to be reassigned from the KSR to either a disk on a different channel, or with a nondefault member name, or card file. The method also requires a preset bootstrap record, but this time the default values for the file and member entries in Table 3-2 can be taken; those entries are PROGFILE and CMDPRC, respectively.

The parameters for the Bootstrap Generator utility program, again assuming a 16K system are:

N,,3FFF,,,,,3480

When you are ready to carry out the loading procedure, press: Stop, Clear, Load, and Execute; after the QLT has executed, again press Execute. The Command Processor indicates its availability by printing a C? on the console; at this point you can use the EX command to assign a command input file that loads CLM with appropriate attachments for configuring your application. No further operator intervention is needed.

LOAD AND HALT PROCEDURES FOR DISK

These procedures can be carried out using either an operator's console or the control panel if no console is available. Both methods are described below.

Loading From Disk With an Operator's Console

When the load device is a disk and an operator's console is available, CLM is loaded using the Command Processor in conjunction with the Disk Loader. The Command Processor accepts control information through the console to establish the environment for the execution of the CLM. (See the Program Development Tools manual for a description of the Command Processor.)

The commands entered through the console specify a relocation factor, and whether a halt should occur after CLM is loaded (e.g., to allow the mounting of a new disk). In addition, the commands specify the command input file name; the overlay file name, if necessary; and the device and channel number from which the CLM commands will be entered.

The last Command Processor command causes the CLM to be loaded. If no halt was specified, the CLM starts executing as soon as it is loaded. Otherwise, the system halts, allowing you to perform any necessary actions before continuing.

Loading From Disk Without an Operator's Console

In this method, no Command Processor is usable. If the load parameters vary from load to load, they can be entered through the control panel, with a bootstrap record on disk set up to halt as described below.

The bootstrap record parameters BTHLT and LDHLT are, in this case, set to Y.

When the bootstrap record has been written on the disk, you can begin the loading procedure. Make sure that your disk is on the device that is connected to the default bootstrap channel (0400_{16}). Press: Stop, Clear, Load, Execute (QLT pause), Execute.

When the 1601 halt occurs, press Stop, and then you can enter the relocation factor into register B2, the HMA into B3, and the loading channel number into R2. Then press Ready and Execute.

When the 1603 halt occurs, you can choose the CLM command input device and channel number by: pressing Stop, entering the channel number of the command input device into R6, and the device type into R7. The device types are: 0040 for a card reader, 0080 for a disk. Press Ready and Execute. Control is now turned over to CLM and configuration of the application proceeds.

LOADING FROM PAPER TAPE

The only procedure available for this medium is a load and halt procedure because the command file for CLM cannot be entered from paper tape. You can minimize halting by setting most values in the bootstrap record, but the LDHLT parameter should be given a value of Y so that you can assign the command file to the appropriate device.

Since paper tape is a serial medium, all elements must appear in the order in which they are to be used. The first element on the tape must be the bootstrap record; when the Bootstrap Generator program is executed to create the bootstrap record, the utility also places the next required module on the tape, namely, the Paper Tape Loader. Table 3-3 shows the order of CLM modules required for paper tape loading. See the Operator's Guide for complete details about all loading procedures.

Table 3-3. CLM Load Module Order for Paper Tape

Memory Size	
HMA=1FFF (8K)	HMA>1FFF (8K)
CLM	CLM
CLM2	CLM2
D$CLMST1	D$CLMST1
D$CLMST2	D$CLMST2
D$COMM[a]	D$COMM[a]
D$CLMFIL[a]	D$CLMFIL[a]
D$CLMBUF[a]	D$CLMBUF[a]
CLMST1[b]	CLMST1
CLMST2	C$CLMST1
COMM[a]	CLMST2
CLMFIL[a]	COMM[a]
CLMBUF[a]	CLMFIL[a]
C$CLMST1	C$COMM[a]
C$COMM[a]	C$CLMFIL[a]
C$CLMFIL[a]	CLMBUF[a]
C$CLMBUF[a]	C$CLMBUF[a]

[a]These modules are included only if the appropriate LACT commands are issued.

[b]Modules beginning with this one are loaded as needed, and in order of LACT command submission when memory size is 8K. The above list assumes that the LACT commands were issued for COMM, CLMFIL, and CLMBUF in that order; consequently, configuration commands should be issued in the same order. Also, for 8K systems, the C$ modules will be loaded after the QUIT command is processed.

If HMA is greater than 8K, all CLM modules will be loaded before any system configuration commands are requested, so there is no necessity to group these commands in this case.

Building a CLM Command File

The order of command submission to the CLM depends on the type of application being configured. If, for example, you are including one or more of the CLM extensions, then the LACT commands are presented first, followed by an ELACT command. If no extensions are included, the ELACT command must be issued so that the CLM can begin to accept system configuration commands.

As to the submission order of the system configuration commands themselves, several factors have an effect upon what the final order should be: memory size in combination with disk availability, the nature of the loading medium, and the characteristics of the modules that make up the application.

As mentioned earlier under "How to Include Optional CLM Extensions," running CLM extensions as overlays is mandatory for 8K systems. Similarly, if memory size is limited, and you are running your own programs as overlays, then the order in which the ADMOD commands are submitted could be important if references are made from one bound unit to another.

The nature of the loading medium can affect the grouping of commands as described previously, under "Loading From Paper Tape."

Finally, the characteristics of the application modules themselves must be considered when you are designing your command file. For example, if you are configuring an application that contains communications software, it is advisable to issue the Communications commands to the CLM early in the process. The reason for this is that the line-type processor modules have extensive initialization code which loads the RAM portion of the MLCP, so that starting the configuration procedure with the Communications commands allows you to use the area occupied by this initialization code for permanent modules loaded later in the process. Whereas, if you wait to bring in the Communications modules until later in the process, you may either waste space, or worse still, you may have to begin the configuration process over again because there was not enough space left for the initialization code to execute.

Similarly, CLM requires space for loading and writing floatable overlays to disk that is usable by permanent code that is subsequently loaded. You should consider loading programs that have floatable overlays early in the configuration process.

Apart from the actual order of the commands in the command file, the following facts should be noted.

Any device that will be accessed through the File Manager requires a DEVFILE command; the DEVFILE command must be issued after the corresponding DEVICE, TTY, BSC, or VIP command.

An error results if the OIM command is omitted from the CLM command file because some system services may use the TYPR facility even if user programs do not.

The CLM command file should begin with a SYS command unless all the default values are taken, and it must end with a QUIT command.

The command set for CLM is described in Table A-1; the definitions for all commands and their parameters are also found in Appendix A.

CLM Action During Loading

When the QUIT command is processed, the data structures are created in a nondedicated area of memory. Figure 3-3 shows how memory looks before the loading phase begins.

When the loader is given control, it obtains the name of the first load module to be loaded from the load module list constructed by CLM. The first module is loaded beginning at a location just above that occupied by the system data structures. After the loading of each module is complete, control is given to the initialization subroutines of the module.

At this point, if the module is a root module with overlays, the overlays are loaded, and written out to the overlay file. Figure 3-4 shows a memory layout after the loading process is completed.

If the space name parameter of the BUFSPACE command is not given a value, the buffer area is obtained from the load residue space, which includes the area from the end of the last load module, through the value of the himem parameter in the SYS command. If the default value of himem is taken, the loader is included in the load residue area and could be overwritten by information put into buffers. Figure 3-5 shows how memory looks during application execution when the value of himem was not changed to protect the loader.

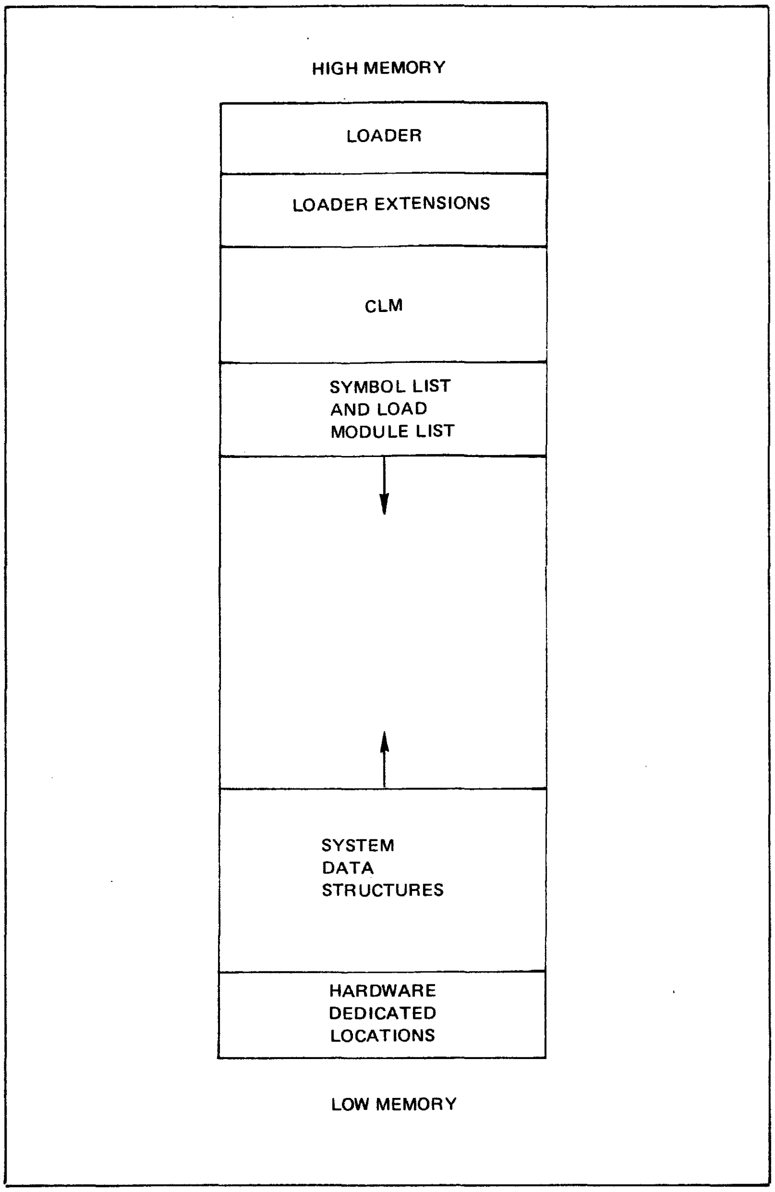


Figure 3-3. Memory Layout During Configuration

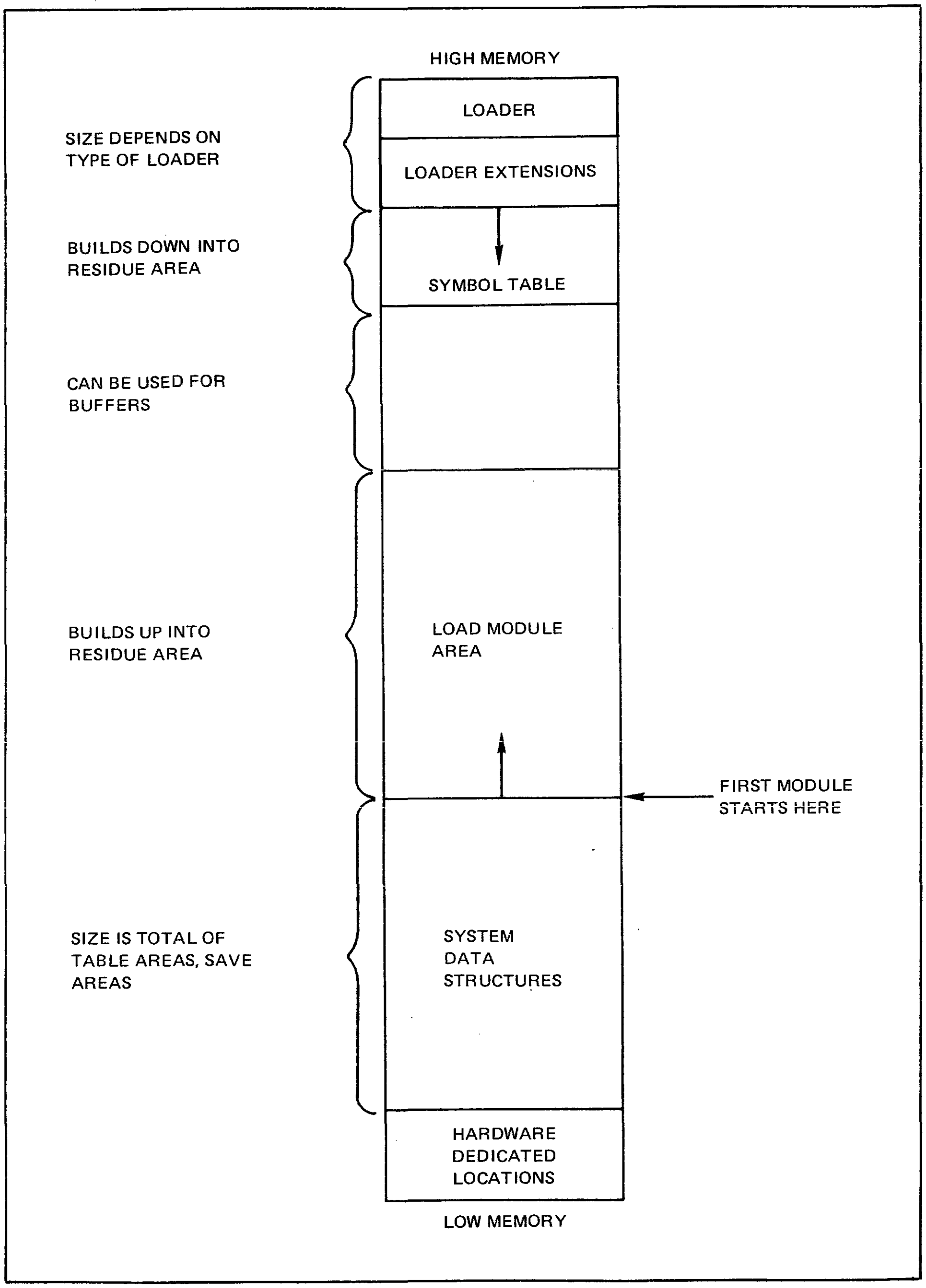


Figure 3-4. Memory Layout After Loading

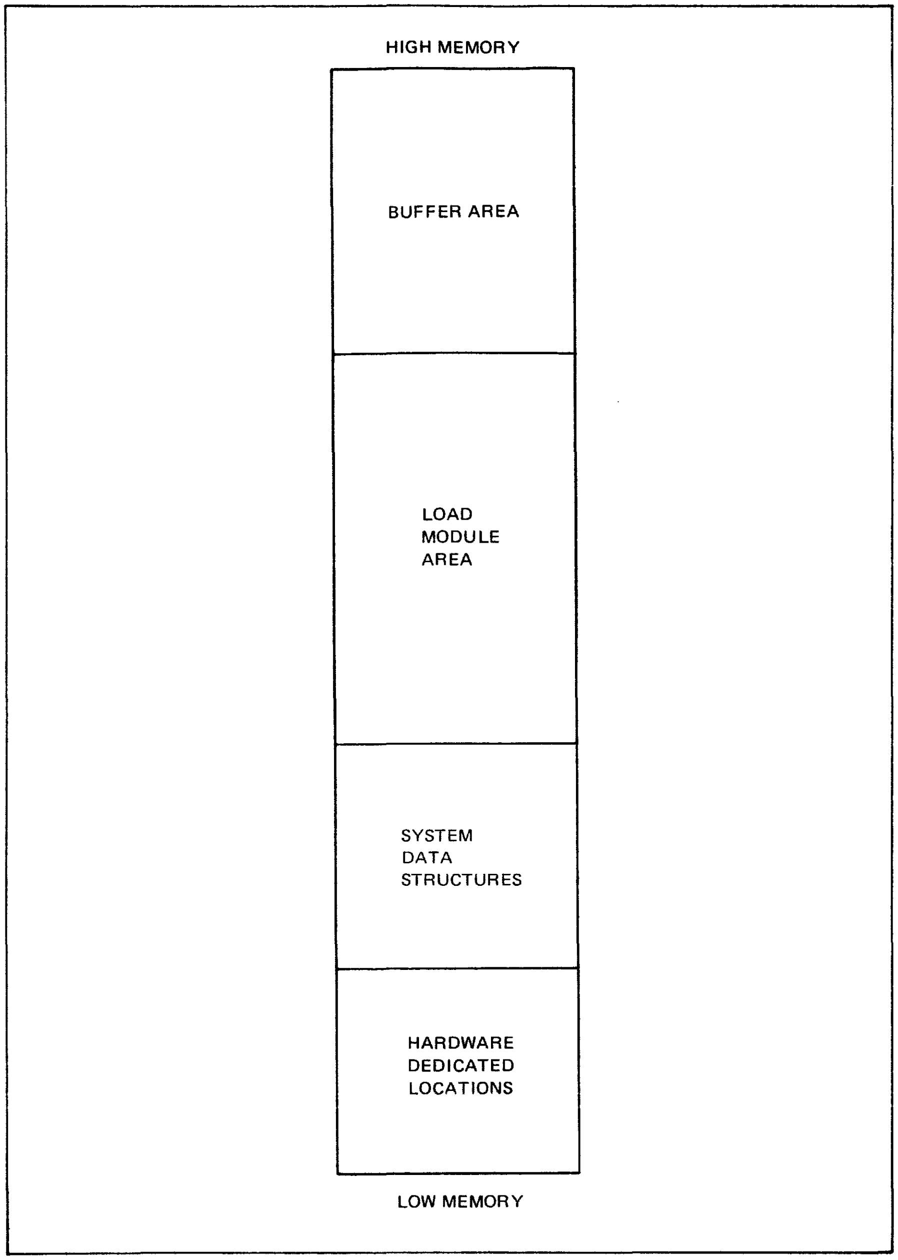


Figure 3-5. Memory Layout During Application Execution

STARTING AN ONLINE APPLICATION

It is important to understand how the starting point of an application is conveyed to the CLM, because you must take specific steps while creating application load modules to ensure that the CLM can identify the desired start address.

Specifically, the CLM TASK command allows a task associated with a priority level to be started at a labeled address when the application is loaded. This start address label must be declared in a Linker EDEF statement when the load module containing the task is created. If an application has more than one task (each on a different priority level) to be started, multiple TASK and EDEF statements can be used. The EDEF's may be in the same or in different load modules.

The EDEF command is used on behalf of assembly, FORTRAN, and COBOL language programs to define start labels to the CLM. Any label may be used in an assembly language program.

For FORTRAN main programs, the label is either:

- The "progname" used in a PROGRAM source statement in the main program, or
- The compiler default label ZFMAIN[1] for a main program that does not contain a PROGRAM source statement.

Note that a PROGRAM statement is required in main programs if multiple start labels are needed.

The rules for defining the start addresses for load modules written in assembly, FORTRAN, and COBOL languages are summarized below.

Assembly Language Start Address Definition

- Start labels chosen are declared by XDEF statements to the Assembler, and EDEF statements to the Linker.
- The label in each TASK command to the CLM matches the XDEF and EDEF definitions.

FORTRAN Language Start Address Definition

- Start labels explicitly declared in PROGRAM source statements (or a ZFMAIN label created implicitly by the compiler) are declared in Linker EDEF statements.
- The label in each TASK command to the CLM matches the EDEF definition.

---

[1]This label is placed into the FORTRAN object (or source) output by the compiler using either an effective (or actual) XDEF Assembler control statement.

- The program name in the PROGRAM-ID clause is declared in a Linker EDEF statement.
- The label in a TASK statement must match the name in the EDEF definition.

If the HLT parameter was coded on the QUIT command to the CLM, a halt will occur after the last load module has been loaded; if not, control will be given to the highest active priority level, and execution begins.

SECTION 4
DEBUGGING

This section provides some practical approaches that may be useful to you when debugging an online application. These suggestions are by no means all-encompassing, nor intended to restrict your ingenuity in uncovering and fixing a software difficulty in your program.

USING THE ONLINE DEBUG PROGRAM

BES provides an interactive debugging component, the Online Debug Program, (ODP) that supplies online patching and testing facilities for application programs running under the BES Executive.

There are two versions of the ODP, one runs as a series of overlays and requires the BES2 Executive; the other is memory resident and can execute under the control of either the BES1 or BES2 Executive. Both versions require an operator's console; an optional, preallocated relative disk file, DEBUG.WORK is used when delayed execution commands are executed. Table 4-1 summarizes the memory and work file space for ODP.

Table 4-1. Memory and Work File Space Usage

Module Name	BES Executive	Memory Needed (Words)	File Space Used Diskette (Sectors)	File Space Used Disk (Sectors)
ZDBG1	ZXEX02 or ZXEX03	2700	22	22
ZDBG	ZXEX03 only	1100 (overlay)	72	47

NOTES:
1. Sector size for diskette is 128 bytes; for disk is 256 bytes.
2. Sector values for the overlay version include required space for two different files: OVERLAY and DEBUG.WORK.
3. Sector values for ZDBG1 and ZDBG represent the optional space provided for the DEBUG.WORK file that is needed only if predefined command lines are to be stored on disk for later execution. (See SF command.)

Online Debug Program Functions

Online Debug Program performs the following functions:

- Defines, stores, and executes (either immediately or after a delay, depending on the command) a sequence of commands from the console, or when breakpoints or trace trap instructions are encountered in the program being tested.
- Sets, clears, or prints breakpoints in task code to monitor task status
- Displays, changes, and dumps either memory or registers; information may be printed on the operator's console, or a line printer
- Evaluates expressions

Debugging Command Language

Commands are submitted to the Online Debug Program through the operator's console or any command terminal. A command line may consist of one or more debugging commands separated by a semicolon and terminated by a carriage return. Some of the commands are executed immediately, and some, by their nature, are executed on a delayed basis. The "predefined" or "delayed execution" commands are stored on disk prior to execution.

Within commands, parameters are separated from one another by one or more spaces. All parameter values are entered using hexadecimal notation.

Any command that produces printed output may direct the output to a device other than the operator's console by using the LRN (logical resource number) of the device; when no LRN is specified, the operator's console receives the output.

Table 4-2 summarizes the debugging commands by function. The following pages present detailed descriptions of the commands and their use.

Table 4-2. Summary of Debugging Commands by Function

Function	Command Mnemonic	Meaning
Command line definition and handling	Dn	Define command line n
	En	Execute command line n
	P*	Print all predefined command lines
	Pn	Print command line n
Breakpoint control	C*	Clear all breakpoints
	Cn	Clear breakpoint n
	GO	Proceed from breakpoint
	L*	List all breakpoints
	Ln	List breakpoint n and associated command line
	Sn	Set breakpoint n

Table 4-2 (cont). Summary of Debugging Commands by Function

Function	Command Mnemonic	Meaning
Trace trap control	DT	Define trace command line
	PT	Print trace command line
Active level control	SL	Set current and active level
	TL	Establish temporary level active
Memory and register control	AR	Print contents of all active level registers
	CH	Change memory
	DH	Display memory in hexadecimal
	DP	Display memory in hexadecimal and ASCII
Symbol control	AS	Assign a hexadecimal value to symbol
	VH	Print value of expression in hexadecimal
General execution	AL	Activate level(s)
	Hn	Print header line
	LL	Line length of console in use
	RF	Reset file location
	SF	Specify file location

DEBUGGING COMMAND FORMAT AND SYMBOLOGY

The format of debugging command lines is:

command-mnemonicΔparamΔparam;command-mnemonicΔparam;. . . ;CR

The symbols in Table 4-3 are used in the command descriptions and examples that appear below.

Table 4-3. Symbols Used in Debugging Command 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 command (DH,CH,DP).
ampersand (&)	Represents the address of the next location beyond the last one used by a previous memory reference command (DH,CH,DP).
brackets [ ]	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.
$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.
G through Z	Twenty single-character temporary symbols having initial values of zero. Values may be assigned using the AS debugging command.
Notational symbols	
braces { }	Indicate optional parameters.
ellipses . . .	Indicates the ability to repeat parameters within braces.
parentheses ( )	Indicate command 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 command definition.
exp	Indicates a valid expression formed using expression elements.
rexp	Consists of exp1/exp2 where exp1 is a hexadecimal number that is a value or a location; exp2 is an optional hexadecimal repeat factor whose value must be between 1 and 32,767. If exp2 is omitted, the value of 1 is assumed.
slash (/)	Brings Online Debug Program to command level; also used to indicate reference to specific LRN for the command use.
Delta (Δ)	Indicates a space.
CR	Indicates a carriage return.
;	Separation character between commands on the same command line.
*	Signifies "all" in the print and clear commands.

Debugging Commands

ACTIVATE LEVEL COMMAND (AL)

Command activates a level corresponding to each expression.

Format:

AL∆exp{∆exp∆. . .} CR

Example:

AL   A   A+2 CR

This example activates priority levels 10 and 12 (decimal)

ALL REGISTERS COMMAND (AR)

Command causes the printing of all registers for the active level.

Format:

AR{/lrn}CR

Example:

AR/3 CR

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. (See Note 6, under "Additional Operating Notes for the Online Debug Program", below)

ASSIGN COMMAND (AS)

Command assigns the hexadecimal value of the expression to the symbol; used to alter registers of the active level, and temporary symbols.

Format:

AS∆sym∆exp{∆sym∆exp ...} CR

Example:

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

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

CLEAR COMMAND (C*)

Command clears all defined breakpoints.

Format:

C* CR

CLEAR COMMAND (Cn)

Command clears specific breakpoint. The value of n may be between 0 and 9.

Format:

CnΔCR

Example:

C3 CR

The example causes breakpoint number 3 to be cleared.

CHANGE MEMORY COMMAND (CH)

Command allows specific memory locations to be given specific values.

Format:

CHΔexpΔrexp{Δrexp...} CR

Examples:

CH 100 0/10 CR

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

CH 200 4FFF CR

Execution of this command puts the value 4FFF into location 200.

CH 2000 0/10 1/10 2/10 CR

This example shows how multiple repeat factors can be used: execution of this command 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.

DEFINE COMMAND (Dn)

Identifies the command line within the parentheses with the number n; n must be a value between 0 and 9.

Format:

DnΔ(command line) CR

Examples:

D3 (CH 100 0) CR

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

This next example illustrates another use of the Dn command:

D4 ( )

namely, command line 4 will be deactivated. When a disk that has predefined command lines from a previous execution is reused, the lines may be referred to without redefinition. (See the Sn command.)

DISPLAY MEMORY COMMAND (DH)

Command causes specified memory locations to be displayed in hexadecimal notation either on the operator's console, or on the device specified by the lrn used.

Format:

DH{/lrn}Δrexp{Δrexp...}CR

Examples:

DH 200 CR

Execution of this command results in the display of the contents of location 200 on the operator's console.

DH/2 200/100 CR

Execution of this command displays the contents of locations 200 to 2FF on the device associated with LRN 2.

DUMP MEMORY COMMAND (DP)

Command causes the display of (a minimum of one line) an area of memory starting at a specified location. Display is in hexadecimal and ASCII notations.

Format:

DP{/lrn}Δrexp{Δrexp...} CR

Examples:

DP 200 CR

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

DP/4 80/40 200/240 CR

This command causes the contents of locations 80 to BF, and 200 to 43F to be displayed on the device associated with LRN 4.

DEFINE TRACE COMMAND (DT)

Command associates the command line within the parentheses with the occurrence of a trace trap or a BRK instruction not already defined as a breakpoint.

Format:

DTΔ(command line) CR

Examples:

DT (AR) CR

This command causes all registers to be displayed each time a trace trap occurs.

This next example illustrates another use of the DT command:

DT ( )

namely, the deactivation of the defined trace command line. When a disk that has predefined command lines from a previous execution is reused, the lines may be referred to without redefinition. (See the Sn command.)

EXECUTE COMMAND (En)

Command executes the predefined command line specified by n, a number from 0 to 9. The En command may not be included in predefined Dn command lines; it is permitted in Sn and trace command lines.

Format:

EN∆CR

Example:

E3 CR

GO COMMAND (GO)

Command results in the resumption of execution on an active level after a breakpoint.

Format:

GO∆CR

PRINT HEADER LINE COMMAND (Hn)

Command causes a header line to be printed; line spacing is controlled by the value of n, such that when n=0, there is a skip to the head of form before the header line is printed; otherwise, the number of lines between 1 and 9 are skipped before printing. A header line may consist of any ASCII characters and/or expressions; expressions are preceded by a percent (%) sign. If a % sign is to be printed, two characters must be used (%%). A header line must end with a space character.

Format:

Hn∆{/lrn}∆(header line) CR

Example:

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

This header will be printed on LRN 2 at the top of a new page as soon as the carriage return is typed. The example illustrates a way to document dumps. The main use of the header command is to document printed information related to breakpoint or trace trap debugging.

LIST ALL BREAKPOINTS COMMAND (L*)

Command lists all breakpoints.

Format:

L*Δ{/lrn}CR

Example:

L* CR

This command will cause all breakpoints to be printed on the operator's console.

LINE LENGTH COMMAND (LL)

Command specifies the line length of the console in use. The length value is expressed in decimal notation, and the limits are: $30 \leq$ value < 126.

Format:

LLΔvalue CR

Example:

LL 72 CR

This command signifies that the console in use has a line length of 72 characters.

LIST BREAKPOINT COMMAND (Ln)

Command causes the listing of a particular breakpoint that was set by a Sn command, and lists the command line.

Format:

LnΔ{/lrn}CR

Example:

L2/4 CR

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

PRINT COMMAND (P*)

Command causes all command lines predefined by Dn commands to be printed.

P*{/lrn}CR

PRINT COMMAND (Pn)

Command causes specified command line predefined by Dn command to be printed.

Format:

Pn{/lrn}CR

The value of n can be between 0 and 9.

PRINT TRACE COMMAND (PT)

Command causes a trace command line to be printed.

Format:

PT{/lrn}CR

RESET FILE COMMAND (RF)

Command resets the location of DEBUG.WORK, and prohibits commands that use this file from operation until another SF command is issued.

Format:

RF CR

SET BREAKPOINT COMMAND (Sn)

Command sets a numbered breakpoint (n) at a particular location. The value of n can be from 0 to 9. When the breakpoint is encountered, an existing command line is executed, otherwise a message is displayed on the console and task execution ceases. The Online Debug Program now has the highest priority. The console message indicates the contents of the location counter and the active priority level.

If there is a preexisting command line associated with a given breakpoint, the old command line must be replaced with a new one, or cleared using empty parentheses ( ); otherwise, the old command line will be executed. (See the Dn command.)

The message format is:

BPn $P=00XXXX $SL=00XX

Format:

SnΔexp {(command line)} CR

Example:

S0 100 (DH 200/10;GO) CR

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

SPECIFY FILE COMMAND (SF)

Command identifies the location of DEBUG.WORK file. Since the function of the command is to open the work file, it should be the first command executed; failure to do this results in the issuing of an error message as soon as a command which requires the work file is used. LRN is specified in hexadecimal notation.

Format:

SFΔLRN CR

Example:

SF B CR

This example indicates the work file to be logical resource number 11 (decimal).

SET LEVEL COMMAND (SL)

Command sets the current and active levels to the value of exp. The current level remains unchanged until another SL command is issued. The default value for the current level is 0.

Format:

SLΔexp CR

Example:

SL C CR

This command sets the level to 12 (decimal).

SET TEMPORARY LEVEL COMMAND (TL)

Command sets the temporary level to the value of exp until another SL, or TL command is issued, or until the end of a command line.

Format:

TLΔexp CR

Example:

TL A;AR;TL B;AR CR

This command causes all registers on levels 10 and 11 to be displayed.

PRINT HEXADECIMAL VALUE COMMAND (VH)

Command prints the value that is the result of the expressions used.

Format:

VH{/lrn}Δexp{Δexp...}CR

Examples:

VH[100]CR

This command causes the display of the contents of the addres found at location 100.

VH .+100-M CR

This command 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.

Using the Online Debugging Program

Program testing and error correction is performed as an interactive dialog between the operator and the Online Debug Program. To achieve control over the task code being tested, the Online Debug Program is given a priority higher than that assigned to task code, but lower than that given to the console and printer used by the operator for the dialog.

The Online Debug Program is included in your application configuration by using the following CLM commands:

```
TSA n,m          (Required if breakpoints or trace traps used.)
EVAL ZDTLRN,TLRN
EVAL ZDDLRN,DLRN
ADMOD filename:ZDBG
TASK ZDTASK,lrn,level,YACT
```

TLRN - The LRN of the command terminal.

DLRN - The LRN of the disk on which DEBUG.WORK file is located.

See Appendix A in this manual for an explanation of the other CLM command parameters.

When the configuration process is finished, the Online Debug Program lets you know that it is ready to accept input by sending a message to the console.

The following example contains typical operations that might be performed in the course of using the Online Debug Program.

Example:

1. Establish header, predefine a command line, initialize a header variable to zero.

   H0(DUMP %M OF AREA 0) CR

   D0 (H0/3;AR/3;DH/3 20/4 [8A]/1A [8B]/1A Y/100) CR

   AS M 0 CR

2. Set breakpoints in code under test.

   S0 300 (AS M M+1) CR

   S1 4A6 (AS M M+1) CR

3. Activate level 8 and wait for breakpoints.

   AL 8 CR

4. When the breakpoint occurs, execute predefined command 0 and then continue.

   E0 CR

   GO CR

NOTE: The predefined command line in the example above (the D0 ...) sets up commands that will display the header, registers, activity indicators, and the ISA's for levels 10 and 11.

ADDITIONAL OPERATING NOTES FOR THE ONLINE DEBUG PROGRAM

1. Online Debug Program can be brought to command level either when the console is idle, or when the ODP is producing output, by typing the "/" character; ODP indicates that it is ready by printing D? at the beginning of a line.

2. If the DP, DH, or VH commands are producing output, and an ! (exclamation mark) is typed, the output will be aborted.

3. Command lines for the Sn, Dn, and DT commands may not exceed 126 characters.

4. A GO command embedded in a breakpoint command line allows task execution to proceed after the desired operations have been performed, without further operator intervention.

5. The following rules should be observed when using breakpoint instructions:

   a. Breakpoints may not be set in code that will be executed at the Inhibit level.

   b. Breakpoints set on the following instructions must be cleared (Cn command) before continuing execution (GO command): any I/O, generic (BRK), scientific, illegal, or LEV instruction, or any instruction with an illegal address syllable.

      Note that the clearing of a breakpoint becomes unnecessary if a second breakpoint command line used on a nonrestricted instruction reinstates the first command line used on a restricted instruction, when both are executed repeatedly within a program loop.

6. Note that the display, change, and dump functions apply to registers on the active level. The active level changes depending on the operation of the Online Debug Program. The active level can be controlled in several ways:

   a. It is set to the level defined by the SL command whenever console input is processed thus allowing the operator to access registers on the same level each time a command is entered from the console, regardless of the change in the active level due to delayed commands since the last command entered from the console.

   b. It is set to the level at which a breakpoint or trace trap occurs, thus allowing the predefined command line being executed to display or alter registers on its own level.

   c. It may be set temporarily with the TL command so that registers of a level different from the active or console levels may be displayed or altered without permanent change to the active or console level definitions.

   d. It is set with the SL command, and this level becomes the console level.

   e. Active level control is designed to assume the value that will most probably be needed based on the ODP action in progress; i.e., console, breakpoint, trace trap, or temporary reference to a different level.

LOCATING LOAD MODULES

The CLM builds data structures, as defined by its commands, and places the load modules immediately following the data structures in memory. Once an application is fully developed, there is no requirement to know the start address of load modules each time the system is loaded because it is invariant and of no concern to the user of the application.

During application development, however, there is a vital need to know where load modules are in memory. Otherwise, the link maps of load modules (based at zero relocation factor) are almost useless. The start address of each load module can be printed on the operator's console by a utility whose load module name is ZXMAP. This information will be of value when debugging, and is displayed under the following two circumstances:

1. Load Phase of CLM reaches the normal halt indication and the Map option of the CLM QUIT command was used to load ZXMAP. The occurrence of other indicators (e.g., multiply-defined or undefined symbols) is incidental. If the application's load modules will fit into available memory on the application development hardware configuration, and there is an operator's console then ZXMAP consists entirely of initialization code. When loaded, (as the last load module), it uses the operator's console channel number known to the loader and prints on the console two sets of information:

   a. Names of undefined symbols

   b. Names and start addresses of all load modules currently memory-resident

2. Load Phase of CLM encounters loader halt, indicating insufficient available memory.

   This circumstance means that the data structures plus the load modules for the specified application will not fit into available memory. ZXMAP may now be used to print on the console a list of load modules that are in memory and those that are not. Although ZXMAP cannot be loaded by CLM since no memory is available, it may be loaded as a stand-alone offline program. The standard bootstrap procedure for offline programs should be used, and the ZXMAP start address (relocation factor) should be specified in low memory to avoid overlaying the CLM information to be printed. The recommended relocation factor for ZXMAP is D0.

   Another possible approach is to load ZXMAP as an ADMOD . . . after selected ADMOD commands of the application. This will produce multiple maps and you can get one indicating memory usage <u>before</u> the CLM attempts to load the module that causes the error halt. This module will be visible in a dump of the loader area of memory at HMA-3 to HMA-12.

   Figure 4-1 shows a sample console output of ZXMAP. It contains the hexadecimal start addresses of the application load modules. FMDEMO is the load module name of the sample user application. To debug FMDEMO the user needs object listings of the modules contained in FMDEMO (e.g., FMA, FMB, FMC) and the link map produced when FMDEMO was created. The link map gives the zero-relative offsets of tags within FMDEMO.

ZXEX02	*0755
ZYFM01	*0DF9
FMDEMO	*14E0
ZIKSR	*16DF
ZILPT	*183E
ZICDR	*18AC
ZIDSK	*1922
ZXMAP	*19CA

Figure 4-1. Sample ZXMAP Output

Assume that FMA, FMB, and FMC were linked in that order and start at relative 0, 100, 200 hexadecimal, respectively. To locate an instruction in memory that is in module FMB, add 100 plus the instruction offset within FMB to $14E0_{16}$. If a load module has not been brought into memory, ZXMAP prints

<load name>*0000

For undefined symbols, ZXMAP prints

<symbol name>*(blank)

DEBUGGING DURING ONLINE APPLICATION DEVELOPMENT

Monitor Points

The system may be monitored in several different ways to verify proper sequencing of memory and/or register contents within routines, or proper task sequencing. Each method requires manual insertion of monitoring code, and implies that space exists within the system for it.

These are ways of creating space to insert monitor points:

- Leave space temporarily in various application modules.
- Append monitoring points using Program Patch.
- Use ODP

The sophistication of the monitoring performed depends on the stage of the application development and testing. The monitoring routine could be a simple halt instruction, a conditional call to the Trace Trap Handler based on current variable status, or an Online Debug Program breakpoint. In each case, you may construct what is required at the time. Program Patch is described in the Utility Programs manual.

If space is allocated in application modules, it may be used by invoking the Online Debug Program.

MANUAL CONTROL

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.

For example:

Source Line	Object Code	
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.

You can use the single word halt effectively in debug phases where only one priority level is active at a time. If more than one level is active, you may use the halt at inhibit level option of the Trace Trap Handler to "freeze" the entire system; i.e., all priority levels including the real time clock. This option is invoked by a BRK generic instruction followed by a HLT instruction. Now use the control panel to enter step mode to examine registers. Then execute a single instruction before restarting program.

Real-Time Clock (RTC)

Some applications require the real-time clock, activated at load time. While the clock is turned on, the CPU is difficult to use in single instruct mode because the RTC is continually generating an interrupt at clock level (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 one instruction from the D0 register as described in the following procedure.

To turn off the clock before starting the application, use the capabilities of executing one instruction from the instruction register (whose selection code is D0) to execute an RTCF instruction while in single instruction mode. Then clear the instruction register (D0), set the P-register (E0) back to CLM halt address and press Ready and Execute. Once the ODP is executing:

a. Press Stop and select D0; change to 0005.
b. Press Execute (this turns off clock)
c. Select D0 and change to 0000.
d. Select E0 and change to CLM halt address.
e. Press Ready and Execute.

Data Structures

There are several useful hardware data structures that contain valuable information for system debugging.

Refer to Figure 4-2 to show portions of an actual memory dump which contains these structures.

```
0010/  0204  0000  0000  0000  0006  0004  0004  0000     HARDWARE DEDICATED LOCATIONS
0018/  0000  0000  0000  0000  0000  0000  0000  001E
0020/  0000  0000  0000  0001  0000  0000  0000  0000

       <------------- ZEROS ------------->

0078/  0000  0000  0000  0000  0000  0000  3830  0000
0080/  0000  0000  0000  0253  018D  01A3  01B9  01CF     ISA POINTERS USED BY THIS APPLICATION
0088/  01E5  01FF  0211  0227  023D  0253  0269  027F

       <------------- ZEROS ------------->

0088/  0000  0000  0000  0000  0000  0000  0000  0187     ISA POINTER FOR LEVEL 63
0BCO   0000  0294  3FFF  02B4  0416  000F  136A  3FFF     CLM-SUPPLIED EXECUTIVE INFORMATION

       <------- CLM DATA STRUCTURES ------->

01B0/  0000  0000  0000  1240  0500  0005  0000  FF00
01B8/  0000  1000  FFFF  0000  07AB  4000  0000  0000
01C0/  1925  0000  31B0  3ED0  3ED3  19D3  19CF  3DA4     ISA FOR LEVEL 6 (BEGINS AT LOCATION 1B9 16)
01C8/  3D64  0000  0000  0000  0000  FF00  0000  13C7
01D0/  FFFF  0000  07A5  4003  183D  02D5  17D5  0C31
01D8/  17D5  02D7  1799  0018  0000  0000  0000  13C0
01E0/  FFFF  0007  0000  FF00  0000  0000  FFFF  0000
01E8/  07AB  4000  0000  0000  183E  0000  0000  0000

       <----- UNUSED INITIALIZED ISA'S ----->

0250/  0000  FF00  0000  0000  FFFF  0000  080B  4003
0258/  0325  033F  10FD  0682  02F5  062F  0135  0014
0260/  FFFF  803F  0050  0000  0050  1000  0000  FF00     ISA FOR LEVEL 13 (BEGINS AT LOCATION 253 16)
0268/  0000  0000  FFFF  0000  07AB  4000  0000  0000

       <------- REMAINDER OF ISA'S ------->

0290/  0000  0000  0000  FF00  0000  0000  0000  0000  -----> SOQ TABLE
0298/  0000  0000  0000  0000  0000  0682  0000  0000
02A0/  0000  0000  0000  0000  0000  0000  0000  0000  -----> EOQ TABLE
02A8/  0000  0000  0000  0000  0000  0682  0000  0000
02B0/  0000  0000  0000  0000  02D5  02E5  02F5  0305  -----> LRT
02B8/  0315  0000  0000  0000  0000  0000  0000  0000
0200/  0000  02D4  0000  0000  02CC  0000  0000  0000

                                      (FOUND BY USING ZXMAP, LINK MAP, AND LISTINGS
                                      AND CLM-SUPPLIED INFORMATION ATCO 16 )
```

Figure 4-2. Hardware/Executive Data Structures

Location 10 contains the trap save area pointer. If traps have occurred, the trap save areas can be located using this pointer. Since CLM creates trap save areas to be contiguous, even unlinked TSA's may be located.

Locations 20 through 23 contain the level activity indicators. They may be examined to discover which level was running or waiting to run. In Figure 4-2, only level 63 (location 23, bit 15) was active.

Location 80 and above contain the ISA pointers for the system. A minimum of 64 locations are always reserved by CLM. Nonzero ISA pointers indicate the potentially-active levels in the system. The pointer in location 83 (level 3) is always either zero or a duplicate of another valid ISA pointer. By matching its contents against the others, you can discover the last level to execute on the inhibit level. In Figure 4-2, it was level 13 (location 8D).

Within the ISA's for each level, the S-, P-, and B5-register entries are of interest. When the S-register contains the level number of the level itself, it was probably interrupted by a higher priority level. If it is zero, the level has probably not been executed. When the level number is 3, the task has terminated using the Task Manager.

The P- and B5-registers may prove useful to determine the starting address of a task or Task Manager entry point.

Device driver ISA's indicate whether the device interrupt has ever occurred. The device places information in the ISA upon interrupt. Figure 4-2 indicates a 13C7 at location 1CF. This is decoded as:

13C0 channel number
07 device level assignment

The level number is the last 6 bits of the 16-bit word.

Software data structures used by the Task Manager and/or device drivers may best be located by finding a reference to them in Honeywell listings, locating those references in the memory dump, and then examining the structures. The information supplied by the CLM immediately after the ISA pointers may be used to find SOQ,EOQ, and LRT (and thereby RCT's) that may be of interest. CLM also provides a pointer to ZXcccc in the Clock Manager module. This is the location immediately before the first of four clock queue data structures. Once these structures are located, enqueued request blocks may be examined and clock timer blocks of drivers analyzed.

Trace History

When using the Online Debug Program with disk-stored command lines that execute upon encountering a trap or a breakpoint, a trace history may be maintained on a line printer. Otherwise, use the Trace Trap Handler.

Handling Load Errors

When the CLM produces a load error (1695) indicating insufficient memory for loading the application, there may be ways to rearrange load modules to solve the problem.

1. Be certain that load modules with large load-time initialization requirements are loaded first (e.g., communications or File Manager) so that maximum advantage is taken of the possibility of overlaying CLM initialization by permanent load modules. Clearly, being unable to load a module because the initialization code is too large can be corrected by placing I/O drivers (DEVICE commands) or the Executive load modules (ADMOD commands) near the end of the load module commands.
2. Be certain that ADMOD commands associated with floatable overlays are also ahead of other modules that do not have overlays or initialization code. Recall that floatable overlays need occupy no permanent memory space outside the CLM residue.
3. Consider recoding nonfloatable overlays as floatable to make better use of CLM residue that can't be used during loading. As an extreme solution for an 8K environment, the bulk of the application could be coded as floatable and called into residue space by its root, which is designed to do nothing more than that.
4. Double check configuration commands to make sure that the minimum amount of space required is used for data structures.

APPENDIX A

CONFIGURATION LOAD MANAGER COMMANDS

Commands entered through the command input stream direct the operation of the Configuration Load Manager (CLM). Using these commands, you can define the environment in which the application is to be run, and specify the modules to be loaded. The configuration commands identify the load module, specify the memory requirements, system services, and peripheral complement to be used; set up internal data structures, and establish trap handling procedures.

Table A-1 summarizes the CLM commands and functions. Individual commands are described in detail later in the section.

Table A-1. Summary of CLM Commands and Command Functions

Command Category	Commands	Functions
System Configuration	SYS, OIM, TSA, TRAP, CLOCK, DATE	Set up data structures in main memory; define application environment.
Load Configuration	ADMOD, ELOC, EVAL	Constructs a load module list consisting of all modules required by the application; list consists of file names with sublists of module names; module names should be unique; the order of modules in the list is critical when they are loaded from a sequential device; permit symbol definition.
Task	DEVICE, TASK, ATLRN, EQLRN	Explicitly specify relationships among tasks, devices, logical resource numbers, and interrupt levels; cause data structures to be built.
Buffer Management[a]	BUFSPACE	Defines buffer pools and related tables that are used by the Buffer Manager.
File Management[a]	FILMGR, DEVFILE, FMDISK, ATFILE	Provide information by which CLM builds the data structures used by File Manager to support a centralized file access capability.

Table A-1 (cont). Summary of CLM Commands and Command Functions

Command Category	Commands	Functions
CLM Control	IOS, *(comment), QUIT	Direct the general actions of CLM; provide documentary information within the command input stream.
Communications[a]	COMM, TTY, VIP, BSC, MODEM, LTPDEF, LTPn, STATION	Explicitly define each communications device; define attributes of communications application; cause data structures to be built; analogous to DEVICE command for peripheral devices; specify relationships among tasks, devices, logical resource numbers, and interrupt levels.
CLM Extensions	LACT, ELACT, IOS	Load the optional extensions to CLM so that file management, buffer management and communications functions can be configured.

[a]Optional extensions that are added to the basic CLM through the use of the LACT and ELACT commands interpret the information supplied by commands in these groups.

COMMAND FORMAT

The CLM accepts commands through the designated command input device. Each command is a separate line of input, consisting of a string of up to 72 ASCII characters. If the command input stream is entered through an operator console device, each command is terminated by a carriage return.

A command line contains a CLM command, including its operands and (optionally) comments. The format of a CLM command is shown below. In this example, lowercase characters indicate items that are to be replaced by actual values. Operands shown within brackets are optional; default values are used if these operands are not specified.

NOTE: The command mnemonic itself need not be specified if all the operands of the command are optional and if the default values of those operands are to be used.

```
Position                                        Position
1                                                     72
mnemonicΔoperand1  [,...,operandn][Δcomments]
```

The command mnemonic, consisting of one or more ASCII characters, specifies the action to be performed. The command mnemonic is separated from its operands by a single space character (indicated by a delta (Δ) character). Commas separate individual operands and a space or carriage return terminates the operand set. Omitted operands are indicated by consecutive commas; trailing commas are not required. Comments can follow the operand field, separated from the operands by one or more space characters. The order of operands in the command line is significant.

The operands associated with the CLM commands can be strings of ASCII characters, or decimal or hexadecimal integers. An ASCII operand begins with an alphabetic character or with an apostrophe character and ends with a comma, space, carriage return, or apostrophe character or when the maximum string length of 64 characters is reached. For purposes of specifying a numeric string, as opposed to a decimal number, the string must be bracketed by apostrophes. ASCII strings are stored in memory in an even number of bytes, left-justified on a word boundary.

An integer used as a command operand is unsigned and can be a single-word decimal or hexadecimal number or a double-word hexadecimal number. The following conventions are used to represent integers in an operand string:

ddddd - Single-word decimal; d is a digit in the range 0 through 9

X'hhhh' - Single-word hexadecimal; h is a digit in the range 0 through F

D'hhhhhhhh' - Double-word hexadecimal; h is a digit in the range 0 through F

In memory, an integer is right-justified on a word boundary, and left-filled with zeros. An operand that specifies an address must be a double-word hexadecimal integer.

NOTE: In the following description of CLM commands, the term "integer" refers to a single-word integer unless otherwise noted.

INPUT DEVICES FOR CLM

Command input to CLM can be submitted on cards, on diskette, or through an operator's console (a KSR device). During the execution of CLM, the command input device can be changed by the IOS command. (See "IOS Command" later in this appendix.)

Under the direction of these commands, CLM accepts load modules on diskette files, loads these modules into memory, and initiates the execution of the application.

The diskette file and member names of the command input to CLM must be CLMCI when the Command Processor is <u>not</u> in use.

ADMOD Command (Add Load Module)

The ADMOD command adds a new module name to the end of the load module list, and specifies that this module is to be loaded during the loading phase. The order of ADMOD commands determines the order in which the load modules are loaded.

NOTE: The fact that new module names are added to the end of the load module list can be significant when loading is to be done from a sequential device, since only one pass is made over the medium.

The format of the ADMOD command is shown below.

ADMODΔfile-name:member-name[,X'channel']

ADMOD - Command mnemonic

file-name - The name of the file in which the load module resides.

member-name - The name of the load module. This load module is a member of the file whose name is specified in the file-name element.

channel - A hexadecimal integer giving the channel number of the device from which the specified file/module is to be loaded. If this operand is omitted, the channel number of the device from which the Configuration Load Manager was loaded is used.

The ADMOD command with the same file-name/member-name as a previous ADMOD command causes the channel number to be updated with the new one. This is useful if a driver module must be loaded from a different channel than the default channel specified in the implicit ADMOD statement that is issued with the following commands: DEVICE, COMM, TTY, VIP and BSC.

The explicit ADMOD command to alter parameters present in an implicitly invoked ADMOD command is issued after the command that caused the implicit command to be issued.

ATFILE Command (Attach File)

The ATFILE command relates a logical file number to a file. The format of the command is shown below.

ATFILEΔlfn,path-name

ATFILE - Command mnemonic.

lfn - The logical file number used to refer to the file once that file has been opened. This value must not exceed the max-lfn specified in the FILMGR command.

path-name - A string of ASCII characters specifying the directory path required to reach the indicated file. The path for diskette begins in the directory of mounted volumes. The path-name has the form: system or user ( ) identifier volume-name>file-name. The greater than symbol (>) must be used to separate the volume-name from the file-name. A volume-name may be up to six characters in length; a file-name may be up to 12 characters long.

A nondiskette path-name has the same form as a diskette file-name; i.e., 1 to 12 characters <u>without</u> the greater than (>) sign preceding it. Refer to the Executive and Input/Output manual, "Glossary of File Manager Terms" for more detail.

<u>ATLRN Command (Attach LRN)</u>

The ATLRN Command relates a logical resource number (LRN) to a physical priority level. The ATLRN command assumes that all levels within the specified range that are not explicitly defined in DEVICE commands are available for use by nondevice tasks.

LRN's not explicitly assigned a level by a DEVICE, TASK, or ATLRN command, remain unassigned. Attempts to use an unassigned LRN result in a "request task" error.

The ATLRN command can also be used to relate additional LRN's to a given level number.

The format of the ATLRN command is shown below.

ATLRNΔlrn,level[,rct-size]

ATLRN - Command mnemonic.

lrn - The logical resource number, no greater than the value of the hilrn operand of the SYS command.

level - The priority level at which the task requested by the specified logical resource number will execute. The value of the level operand cannot be less than 5 nor greater than the value specified as the lolevel of the SYS command.

rct-size - An integer that gives the size, in words, of the RCT for the associated LRN. The default value is one word. If the rct-size parameter is omitted, the default value will be assumed, unless the level parameter is the same as that in a previous DEVICE, TASK, TTY, VIP, BSC, LTP, STATION, or ATLRN command; in this case, no new RCT is created, the lrn becomes a synonym for the lrn in the previous command having the same level parameter value.

The following rules apply to an ATLRN command:

- An ATLRN command with an rct-size parameter always produces an RCT of that size; its LRN is never a synonym.
- The lrn of an ATLRN command that has no rct-size parameter is synonymous with the lrn of the immediately previous TASK, DEVICE, TTY, VIP, BSC, LTP, STATION, or ATLRN command having the same level parameter value.
- The default value of the rct-size parameter is one word.

The use of the ATLRN command allows you to relate RCT's of different sizes to the same priority level. Consider the following set of commands:

```
DEVICE CDR,1,6,X'0580'
ATLRN 2,6
ATLRN 3,6,19
ATLRN 4,6,37
ATLRN 5,6
```

The RCT constructed as a result of a DEVICE command is 16 words long; LRN 2 is a synonym for LRN 1 and refers to the same RCT. LRN's 3 and 4 are unique, and refer to RCT's of 19 and 37 words, respectively. LRN 5 is a synonym for LRN 4, and refers to the same RCT. Priority level 6 then, has three RCT's associated with it: one of 16, one of 19, and one of 37 words.

BSC

BSC 2780 Command

This command identifies each binary synchronous communications line included in the system. The format of the BSC command is:

BSCΔlrn,level,channel[,label][,modem][,primary/secondary][,character-set]

BSC - Command mnemonic.

lrn - The logical resource number by which the device is requested. It must be less than or equal to the hilrn parameter of the SYS command.

level - The priority level at which the driver for the device will execute. Must be less than or equal to the lolevel parameter of the SYS command; it may be the same as other communications devices, but must be different from the level specified in the COMM command, and from the levels for noncommunications tasks and devices.

channel - The channel number of the device.

label - A label, assumed to be a location definition, which must be defined in a load module. This label is the entry point of the attention subroutine. The default is null.

modem - A number specifying the type of data set. Possible values are:

0 - Direct connect.

1 - Bell 1xx-type modem (103A,113F,etc). Both data-set-ready and carrier-detect signals are needed for a connection; lack of both signals is a disconnection.

2 - Bell 2xx-type modem (201A,201C,208A,etc). The data-set-ready signal is needed for a connection; lack of this signal is disconnection.

3 or greater - User-defined modem type (see MODEM command).

The default value is modem type 2.

primary/secondary - Values may be specified as P or S; indicates whether this is the primary or secondary endpoint of the transmission. A primary endpoint has higher priority when sending data.

character set - One of the following may be specified:

AS - ASCII (default)

EB - EBCDIC

TE - Transparent EBCDIC

When this command is processed, an implicit ADMOD command is issued to include the BSC line-type processor (ZQPBSC) in the load list.

BUFSPACE

BUFSPACE Command (Pool Definitions)

The BUFSPACE command defines contiguous areas of memory (called "blocks") to be used as buffer areas. Blocks of uniform size are linked into a pool. Each pool is controlled by a pool parameter block (PPB), which describes the location of the first block in the pool and the size of blocks in this pool. A set of PPB's, in order by block size, forms a pool parameter table (PPT). The information contained in the PPT is required if the Buffer Manager function of the Executive is used. The BUFSPACE command can be used to:

- Assign a label to the start of the PPT
- Specify the size of each block and the number of blocks in each pool
- (Optionally) Designate a predefined label in an existing load module as the start of the memory area containing the buffer pools. Alternatively, buffers are created in the residual memory area between the end of the last load module and the high memory address specified in the SYS command.

The BUFSPACE command defines one PPT and its associated buffer pools. The CLM arbitrarily creates the PPT in nondedicated memory and defines the value of the ppt-label parameter as the start of the PPT.

The format of the BUFSPACE command is shown below.

```
BUFSPACEΔppt-label,[space-name],size,number[,size,number,...]
```

ppt-label - The label assigned to the first word of the pool parameter table.

space-name - The label of the beginning of a contiguous area in main memory, large enough to contain all the pools defined by the succeeding operands in this command. If this operand is omitted, the buffers will be built in residual main memory space, between the end of the last load module and the high memory address.

size,number - A pair of integers specifying the size (in words) of each block and the number of blocks in one buffer pool. As many size, number operand pairs as are needed can be specified.

If the entire BUFSPACE command cannot be included on one line, additional BUFSPACE commands may be issued having the same ppt-label, a null space-name operand, and additional size, number operand pairs.

An operand pair with the same size parameter as a previous one (the same ppt-label), causes the new number to replace the old one.

The same space-name in an additional BUFSPACE command as in a previous one results in an error condition.

A BUFSPACE command with a nonnull space-name, and a ppt-label the same as a previous one, replaces the old space-name with the new one.

CLOCK

CLOCK Command (System Clock)

The CLOCK command specifies the line frequency used to drive the system clock, and the period between clock-generated interrupts (i.e., the timeout interval). The format of the CLOCK command is:

CLOCKΔ[hz],[scan-cycle]

CLOCK - Command mnemonic.

hz - Line frequency. Possible values are 50 to 60. The default value is 60 (U.S. standard).

scan-cycle - The time, in milliseconds, between periodic real-time clock-generated interrupts. The default value is 50 milliseconds. The following lists show the possible values of the scan-cycle for both line frequencies:

50-Hz line (milliseconds)	60-Hz line (milliseconds)
10	8
20	16
50	25
100	33
	50
	100

COMM (Communications System Command)

This command specifies the interrupt priority level for all communications devices. This level should be higher than all other devices and tasks, the recommended level is 5. The format of the command is:

COMMΔlevel

COMM - Command mnemonic.

level - The priority level used as an interrupt level for all communications devices. Must be greater than or equal to 5, and less than or equal to the lolevel parameter of the SYS command, and must be unique. The COMM command must precede all other CLM communications commands.

When this command is processed, two implicit ADMOD commands are issued: one for the Communications Supervisor, and one for the MLCP Driver. The default channel number (from which the CLM was loaded) is assumed. If necessary, explicit ADMOD commands, issued after the command that caused the implicit command to be issued, can be used to change the channel number. The implicit commands are:

```
ADMOD CLMCOMM:ZQEXEC (For the Communications Supervisor)
ADMOD CLMCOMM:ZQMLON (For the MLCP Driver)
```


DATE Command (Date and Time)

The DATE command specifies the current date and time. The format is:

DATEΔ['yymmdd'][,'time']

DATE - Command mnemonic.

yymmdd - An ASCII numeric string providing the current year, month, and date. If this operand is omitted, the default value is null.

time - An ASCII numeric string providing the time of day, in the format hhmm, where hh is the hour of the day (an integer in the range 00 to 23), and mm is the minute of the hour (an integer in the range 00 to 59). If this operand is omitted, the default value is null.

DEVFILE Command (File Management Devices)

The DEVFILE command identifies the nondisk devices that can be used by the File Manager. For any given device, the DEVFILE command must be issued after the corresponding DEVICE, TTY, VIP, or BSC command that defines its device type and logical resource number. The format of the DEVFILE command is shown below.

```
DEVFILEΔdevice-name,lrn,file-name[,double][,share][,rec-size]
```

DEVFILE - Command mnemonic.

device-name - A string of ASCII characters identifying the device. Possible values for the DEVFILE (column 2, below) are:

Device Type	DEVFILE Command	DEVICE Command
KSR - input and output	KSR	KSR
KSR - input only	KSI	KSR
KSR - output only	KSO	KSR
ASR - keyboard input/output	ASR	ASR
ASR - keyboard input only	ASI	ASR
ASR - keyboard output only	ASO	ASR
ASR - paper tape reader	TTR	ASR
ASR - paper tape punch	TTP	ASR
Line printer	LPT	LPT
Serial printer	SPT	SPT
Card reader	CDR	CDR
Diskette	(See FMDISK)	DSK
Cartridge disk (removable)	(see FMDISK)	RCD
Cartridge disk (fixed)	(See FMDISK)	FCD
TTY - input and output	TTY	(See TTY command)
TTY - input only	TTYI	
TTY - output only	TTYO	
VIP - input and output	VIP	(see VIP command)
VIP - input only	VIPI	
VIP - output only	VIPO	
BSC - input and output	BSC	(see BSC command)

Note that the corresponding device-name (column 3, above) must have appeared in a previous DEVICE, TTY, VIP or BSC command.

lrn - Logical resource number by which the device is requested. The value of this operand must not exceed the value of the hilrn operand specified in the SYS command. The operand must have appeared in a previous DEVICE, TTY, VIP or BSC command.

file-name - A string of up to 12 ASCII characters specifying the name by which the file (device) is identified within the application.

double - If the ASCII character D is specified for this operand, all reads and writes to this file will be double buffered. If the operand is omitted, file reads and writes are not double buffered.

share - If the ASCII character S is specified for this operand, the device can be shared. If the operand is omitted, the device cannot be shared.

rec-size - The maximum record size in bytes for the device file described in this command. The default (decimal) values for individual devices are:

KSR/ASR/TTY	72
ASR (read or punch)	32,767
VIP (input and output)	32,767
VIP (input or output only)	80
BSC (input and output)	32,767
Line printer	137
Serial printer	133
Card reader	80

NOTES:
1. The "double" and "share" parameters are mutually exclusive, you cannot use double buffering with a shared file.
2. A file cannot be bidirectional and double buffered.
3. Double buffering should be used in conjunction with the following device-name parameters: KSI, KSO, TTYI, TTYO, VIPI and VIPO.

DEVICE

DEVICE Command (I/O Device Task)

Each device to be used in the application must be explicitly defined in a DEVICE command. In addition, the DEVICE command implicitly defines the load module for the driver. The device-type operand is a generic name - there may be more than one device of the same type in the application; e.g., two diskettes. The level and channel operands, however, must be unique for each device. Device levels usually occupy a higher priority than task levels. The lrn operand specifies the logical resource number for the device. The lrn by which a device is requested need not be unique, but if a device is requested by more than one lrn, the ATLRN command must be used to relate these additional lrn's to the single level for that device.

Specifying the same device-type and lrn operand values in more than one DEVICE command causes the previous DEVICE command to be updated with the new level and channel operand values.

The format of the DEVICE command is shown below.

DEVICEΔdevice-type,lrn,level,channel[,label]

DEVICE - Command mnemonic.

device-type - A string of ASCII characters identifying the type of device. Possible values and associated devices are shown below.

Device Type	Operand Value	Driver Name
KSR	KSR	ZIKSR
ASR	ASR	ZIASR
Line Printer	LPT	ZILPT
Serial Printer	SPT	ZILPT
Card Reader	CDR	ZICDR
Diskette	DSK	ZIDSK
Cartridge disk	FCD (fixed)	ZICDSK
Cartridge disk	RCD (removable)	ZICDSK

lrn -The logical resource number by which the device is requested. The value of this operand must be an integer that is less than or equal to the hilrn value specified in the SYS command.

level - The priority level at which the driver task for the device will execute. The value of the level operand cannot be less than 5 nor greater than the value specified as the lolevel parameter of the SYS command.

channel - The channel number of the device.

label - The label parameter may be either an ASCII value (location definition), or an integer (reference LRN) that is the lrn parameter of a previous DEVICE command. When the label parameter is given an ASCII value, the value must be defined in a load module. The address of the label is stored as the first entry of the device-specific words in the device resource control table (see the Executive and Input/Output manual). This parameter can be used by the drivers as needed, except that when it appears in a DEVICE command describing a KSR or an ASR, it must be the entry point of the attention subroutine. The default label for LRN 0 (operator's console) is ZIATTN, which is defined in the Executive load module. The default for other LRN's is null.

When the label parameter is used as a reference LRN (i.e., the value is identical to the LRN value of a previous DEVICE command), the location of the RCT for the previously defined device is stored as the first entry of the device-specific words in the RCT for the currently defined device. Conversely, the location of the RCT for the currently defined device is stored in the corresponding position in the RCT of the previously defined device. DEVICE commands specifying removable and fixed cartridge disk devices must cross-reference each other in this manner.

The following pair of DEVICE commands illustrates a valid use of the label parameter as a reference LRN:

```
DEVICE FCD,6,10,X'1280'
DEVICE RCD,9,10,X'1280',6
```

The following rules apply to the use of the label parameter as a reference LRN:

- The first DEVICE command of a related pair may not have a reference LRN to another DEVICE command; result is an error.
- The level and channel parameters of a DEVICE command that has a reference LRN must be the same as those in the related DEVICE command.
- Given a related pair of DEVICE commands, the reference LRN must be the same as the LRN in the related DEVICE command.

An implicit ADMOD command for the driver load module of the form:

ADMOD CLMFILE:<driver-name>

is issued with each DEVICE command. The default channel number is assumed. If the default channel number cannot be used, an explicit ADMOD command having the file-name:module-name but a new channel number can be used to change the channel number. (See the ADMOD command.)

An implicit TASK command of the form:

TASK start-address(of the device),lrn,level

is also issued with each DEVICE command.

ELACT

ELACT Command (End Load Action)

The ELACT command indicates that all interpretive modules have been included, and that all commands submitted to the CLM can be processed. The format of the ELACT command is:

ELACT

ELACT - Command mnemonic.

NOTE: Prior to the processing of the ELACT command, only the IOS, LACT, and ELACT will be recognized as valid commands, the submission of any other command will result in an error condition. Once this command is processed, all other commands will be accepted, and the IOS, LACT, and ELACT commands will be invalid.

The ELACT command must be issued even if no optional modules are to be added to CLM.

ELOC Command (Define Address Symbol)

The ELOC command defines a symbolic name as an absolute address. The definition is stored in the symbol table, and redefinition is not allowed. The symbolic name may be referred to in the loading process. The format of the ELOC command is shown below.

ELOCΔsymbol,D'absolute-address'

ELOC - Command mnemonic.

symbol - One through six ASCII characters specifying the symbolic name to be assigned.

absolute-address - The double-word hexadecimal integer specifying the absolute address that is the definition of the symbol.

EQLRN Command (Equate LRN's)

The EQLRN command provides for the definition of LRN synonyms. The format is:

EQLRNΔnew-lrn,old-lrn

EQLRN - Command mnemonic.

new-lrn - A integer, no greater than the value of the hilrn parameter of the SYS command, that is to be equated to a previous LRN.

old-lrn - The value of a previously assigned LRN for which a synonym is being provided.

EVAL Command (Define Value Symbol)

The EVAL command defines a symbolic name as a value. The definition is stored in the symbol table, and redefinition is not allowed. The symbolic name may be referred to during the loading process. The format of the EVAL command is shown below.

EVALΔsymbol,value

EVAL - Command mnemonic.

symbol - One through six ASCII characters specifying the symbolic name being defined.

value - A single-word integer whose value becomes the definition of the symbol.

FILMGR Command (File Manager)

The FILMGR command defines the general File Manager variables. This command must precede any other file management commands. The format of the FILMGR command is shown below.

FILMGRΔ[max-lfn][,concurrentcalls][,concurrent opens]

FILMGR - Command mnemonic.

max-lfn - A value ≤255 representing the highest logical file number (LFN) permitted in the application. The LFN is the value used to refer to a file once that file has been opened. The default value of this operand is 15.

concurrent calls - The number of concurrent calls to the File Manager. This number must be an integer greater than zero. The default value is 4.

NOTE: Each task can have only one call to the File Manager at a time, but a number of tasks can have one call each at a given point in time.

concurrent opens - The number of concurrently open files. This number must be an integer greater than zero. The default value is 8.

FMDISK Command (File Management Disk)

The FMDISK command identifies the disk devices available to the File Manager. The format of the FMDISK command is shown below.

FMDISKΔdisk-type,lrn

FMDISK - Command mnemonic.

disk-type - Specifies the disk device.

DSK - Diskette
RCD - Removable cartridge disk
FCD - Fixed cartridge disk

lrn - The logical resource number by which the device is requested. The value of this operand must not exceed the value of the hilrn operand specified in the SYS command.

IOS Command (I/O Stream)

Using the IOS command, you can change the command input stream from one device to another. The format of the IOS command is shown below.

IOSΔCI$,device,X'channel'[,member-name]

IOS - Command mnemonic.

CI$ - The name of the command input stream.

device - A string of ASCII characters designating the new command input device. Possible values are: device mnemonics ($CDR or $KSR) or a disk file-name.

channel - A hexadecimal integer specifying the channel number of the new command input device.

member-name - The member name of the command input list on a disk device. The file in which this member resides is the file-name parameter. If the parameter is omitted, CLMCI is assumed.

LACT

LACT Command (Load Action)

The LACT command identifies a load module to be added to CLM in order to provide interpretation of one of the supplementary command groups. One LACT command must be included for each set of command group extentions required in the configuration. The format of the LACT command is:

LACTΔfile-name:module-name[,X'channel'][,waid][,overlay]

LACT - Command mnemonic.

file-name - The name of the file in which the interpretive modules for the particular command group reside.

member-name - The member name of the load module that provides the interpretive routines for the particular command group.

channel - The channel number (hexadecimal) of the device from which the load module is to be loaded. The default value is the channel from which the CLM was loaded.

waid - The identification number of the work area for this module. If this number is omitted, the work area is not to be shared; if the number is the same as that supplied in the LACT command for another interpretive module, the work area can be shared.

overlay - The letter O indicates that the interpretive modules specified in this command are to be treated as overlays during the execution of CLM; if the parameter is omitted, all modules are resident in main memory during CLM operation. The parameter must be coded when HMA is 1FFF (8K).

LTP

LTPDEF (Command (LTP Definition)

This command specifies the size of the communications tables that the user-written LTP requires. The command is optional, but if used, must precede the LTPn command that refers to it. The format is:

LTPDEFΔn,channel-table-size,station-table-size

LTPDEF - Command mnemonic.

n - Specifies which LTP is being defined; n is a number from 0 to 3.

channel-table-size - Specifies the number of words needed for the channel table and the CQB's. The default value is 32 words.

station-table-size - Specifies the number of words needed for this LTP's station table (RCT). The default value is 7 words.

LTPn

LTPn Command

This command specifies the characteristics of a nonstandard communications device. For each device driven by a user-written LTP, this command must be issued. The format of the command is:

LTPΔlrn,level,channel[,label][,modem][,speed][,FDX/HDR][,LTP-specific-word]

LTPn - Command mnemonic. There may be up to four user-written LTP's included in a configuration. The mnemonics could be: LTP0, LTP1, LTP2, or LTP3, depending on which user-written LTP is being defined. CLM saves the number in the channel table for the device for use by the LTP's initialization code.

lrn - Specifies the logical resource number by which the device is requested; must be less than or equal to the hilrn parameter of the SYS command.

level - The priority level at which the driver for the device will execute. Must be less than or equal to the lolevel parameter of the SYS command; it may be the same as other communications devices, but must be different from the level parameter in the COMM command, and from the levels specified for noncommunications tasks and devices.

channel - The channel number of the device.

label - A label, assumed to be a location definition, which must be defined in a load module. This label is the entry point of the attention subroutine. The default label for LRN 0 (operator's console) is ZIATTN, which is defined in the executive load module. The default for other LRN's is null.

modem - A number specifying the type of data set. Possible values are:

0 - direct connect

1 - Bell 1xx-type modem (103A,113F, etc.) Both data-set-ready and carrier-detect signals are needed for a connection; lack of both signals is a disconnection.

2 - Bell 2xx-type modem (201A,201C,208A, etc.) The data-set-ready signal is needed for a connection; lack of this signal is a disconnection.

3 or greater - User-defined modem type. (See MODEM command.)

The default value is modem type 2.

NOTE: If the line is direct connect and asynchronous, modem type 2 must be specified; if the line is direct connect and synchronous, specify modem type 0.

speed - The data rate in bits per second. The default value is zero, and signifies a synchronous line. One of the following values must be specified for an asynchronous line:

50	300	2400
75	600	3600
110	900	4800
134.5	1200	7200
150	1800	9600

FDX/HDX - Specifies whether the procedure is full or half duplex. If it is full duplex (FDX), two channel tables will be assigned. The default value is HDX.

LTP-specific-word - A word containing user-defined information to be passed to the LTP via the station table at offset ZQSSTS. The default is zero.

NOTES: 1. An LTPDEF command must precede its corresponding LTPn command unless default values are to be taken for the channel and station table sizes.

2. Each LTP load module must be added to the load module list constructed by CLM in the usual way; i.e., by being identified in an ADMOD command.

MODEM

MODEM Definition Command

This command is used to define an nonstandard modem type. (See the MLCP Programmer's Reference manual for details about contents of the line control tables.) The information provided in this command is used to test entries in the LCT for the device to verify a connection or a disconnection. The format of the MODEM command is:

MODEMΔtype-number,connection-AND-mask,connection-XOR-mask,
disconnection-AND-mask,disconnection-XOR-mask,data-set-control

MODEM - Command mnemonic

type-number - An integer from 3 to 15 that is assigned to this modem definition and may then be used in a communications device command.

connection-AND-mask - A 2-digit hexadecimal number whose value determines which bits of LCT entries 14 (receive channel data set status) and 46 (transmit channel data set status) will be examined when a connect request is processed.

connection-XOR-mask - A 2-digit hexadecimal number whose value specifies which bits of LCT entries 14 and 46 must be on for a connection

disconnection-AND-mask - A 2-digit hexadecimal number whose value determines which bits of LCT entries 14 and 46 will be examined when a disconnect request is processed, or when a test for the occurrence of a disconnect is made.

disconnection-XOR-mask - A 2-digit hexadecimal number whose value determines which bits of LCT entires 14 and 46 must be on for a disconnection. (Entries 14 and 46 of the LCT are the data set status for the receive and transmit channels, respectively.)

data-set-control - A 2-digit hexadecimal number placed in entry number 20 of the LCT and line register 2 (LR2) of the communication line adapter (CLA) when a line is to be connected.

NOTES: 1. To test for a successful connection, entries 14 and 46 of the LCT are first subjected to a logical AND operation against the (user-supplied) connection-AND-mask; then a logical exclusive OR operation is performed on the result of the first operation, against the (user-supplied) connection-XOR-mask. If the result is zero, a connection has been established.

2. To test for a disconnection, the same operations are carried out using the analogous disconnection masks. A zero result indicates a disconnection.

3. The following table shows the mask and data set control values for the standard, CLM-recognized modem types:

Modem Type	Type Number	Connection Masks		Disconnection Masks		Data Set Control
		AND	XOR	AND	XOR	
Direct Connect	0	X'80'	X'80'	X'80'	X'00'	X'88'
Bell 1xx	1	X'A0'	X'A0'	X'A0'	X'00'	X'80'
Bell 2xx	2	X'80'	X'80'	X'80'	X'00'	X'80'

OIM

OIM Command (Operator Interface Manager Definition)

The OIM command defines the lrn and level required by the Operator Interface Manager. This command must be present, or an initialization error will occur during the loading of the Executive load module. The format of the OIM command is:

OIMΔlrn,level

OIM - Command mnemonic.

lrn - The logical resource number reserved for use by the Operator Interface Manager. The value is an integer that is less than or equal to the value of the hilrn parameter in the SYS command.

level - The priority level at which the Operator Interface Manager operates. The value cannot be less than 5 nor greater than the lolevel parameter of the SYS command.

QUIT

QUIT Command (Initiate Loading)

The QUIT command is the last configuration command in the command input stream. When this command is encountered, the CLM stops reading commands from the command input file and initiates the loading phase. As a last step in the configuration phase, the CLM creates a set of nondedicated data structures (tables, save areas, pointers, etc.), based upon the information contained in the previous commands.

If the HLT parameter is present in the QUIT command, the processor will halt after the configuration loading is completed and before the application is started. The format of the QUIT command is shown below.

QUITΔ[HLT][MAP]

QUIT - Command mnemonic.

HLT - Specification of this parameter causes the machine to halt after loading and before beginning the execution of the application. The default assumption is not to halt. Do not execute a control panel master clear operation before application execution.

MAP - Specifying this parameter causes ZXMAP to be loaded last (provided an operator console is present) automatically. ZXMAP must be on the same file as CLM.

This parameter is equivalent to an implicit ADMOD command:

ADMOD PROGFILE:ZXMAP

STATION

STATION Command

This command is used to specify additional devices on lines controlled by LTP's that have been written to handle multiple devices per line. One device on the line must be identified by describing it in an LTPn command; additional devices are specified in STATION commands, one per device, immediately following their corresponding LTPn commands. The format of the command is:

STATIONΔlrn[,label][,LTP-specific-word]

STATION - Command mnemonic.

lrn - Specifies the logical resource number by which the device is requested; must be less than or equal to the hilrn parameter of the SYS command.

label - A label, assumed to be a location definition, which must be defined in a load module. This label is the entry point to the attention subroutine. The default label for LRN 0 (operator's console) is ZIATTN, which is defined in the executive load module. The default for other LRN's is null.

LTP-specific-word - A word containing user-defined information to be passed to the LTP via the station table at offset ZQSSTS.

NOTE: The priority level, channel number, modem type, line speed, and line procedure (FDX/HDX) of devices described in STATION commands, are obtained from the preceding LTPn command.

SYS

SYS Command (System)

The SYS command defines the environment in which the online application will be run. When specified, the SYS command must be the first CLM command entered (with the exception of the IOS or DATE command). The format of the SYS command is:

SYSΔ[,hilrn][,lolevel][,SAF][,D'himem']

SYS - Command mnemonic.

hilrn - The highest logical resource number (LRN) to be used by the application. The specification of this operand determines the size of the logical resource table (LRT) for the application. (The size of the LRT equals hilrn+1.) The default value of hilrn is 15. The maximum value is 255.

lolevel - The lowest priority level to be used by the application. This parameter also establishes the number of interrupt save areas (ISA's) and affects the total area set aside for ISA's and for the start-of-queue and end-of-queue header tables. The value of the lolevel operand must be between 6 and 62 inclusive. The default value is 15.

SAF - Model designator. The default value is SAF.

D'himem' - The double-word hexadecimal integer specifying a main memory address. The himem operand permits a main memory area between the end of the system and the physical end of memory to be used for nonsystem use (e.g., for storing an offline dump routine). The default value of the himem operand is the high-memory address of the loader. It is the end of the main memory area for buffers.

TASK

TASK Command (Define Task)

The TASK command defines an initial start address for a level. It is used for a task that requires exclusive use of a level (i.e., is requested by means of an implicit-start-address request block) or by an initially active application task. The format of the TASK command is:

```
TASK start-address,lrn,level[,activity]
```

TASK - Command mnemonic.

start-address - An ASCII label that is the start address of the first task code to execute on a particular level after the system is started. The label is declared in an XDEF statement in an assembly language program, and an EDEF statement to the Linker.

lrn - The logical resource number by which the task is requested. The value of this operand must be an integer no greater than the hilrn value specified in the SYS command.

level - The priority level at which the task will execute. The value of the level operand cannot be less than 5 or greater than the value specified as the lolevel of the SYS command.

activity - The value of this operand determines the setting of the level activity indicator. Possible values and their interpretations are:

YACT - The task is initially active.

NACT - The task is not initially active.

The default value is NACT. It is the task on the highest priority level that has been declared active (by a YACT in its TASK command) that is entered when execution starts.

TRAP

TRAP Command (Trap Vector)

The TRAP command establishes a relationship between a trap vector number and a trap handler name. During execution, trap handling procedures are activated only if the appropriate TRAP command has been specified. The three Honeywell-supplied trap handlers are the Trace Trap Handler, associated with trap vector 2, the Floating-Point Simulator, associated with trap vector 3, and the Scientific Branch Simulator associated with trap vector 5.

If the TRAP command is used, the trap handler must be in a load module to be loaded. Furthermore, the label of its entry point (i.e., the handler name) must be declared at link time with the EDEF Linker command. If more than one TRAP command is issued for the same trap vector number, the last TRAP command overrides all previous ones for the same trap number. There are no default values for this command. The command format is:

TRAPΔtrap-number,handler-name

TRAP - Command mnemonic.

trap-number - The number of the trap vector between 1 and 46.

handler-name - A string of ASCII characters specifying the name (label) of the start of the trap handler. This label must be defined in the load module for this application. The three Honeywell-supplied trap handlers have the following names:

1. ZXTRAC - Trace Trap Handler
2. ZFPSIM - Floating-Point Simulator
3. ZFBSIM - Scientific Branch Simulator

TSA

TSA Command (Trap Save Area Definition)

The TSA command defines the number and size of the items in the trap save area (TSA) list. When a trap occurs, certain pertinent information is stored in a trap save area item in main memory. The TSA command allows the adjustment of the number and size of these items for optimum memory usage. All items in the TSA list are of the same size. To find the total size of the trap save area, multiply the number of items by the size of each item. The format of the TSA command is shown below.

TSAΔ[,number of items][,size]

TSA - Command mnemonic.

items - The number of trap save area items required. The default (and the minimum) value is 2.

size - The size (in words) of one trap save area item. The default (and the minimum) value is 8.

TTY

TTY Command

This command identifies each teleprinter device in the application. The format of the TTY command is:

TTYΔlrn,level,channel[,label][,modem][,speed]

TTY - Command mnemonic.

lrn - The logical resource number by which the device is requested. It must be less than or equal to the hilrn parameter of the SYS command.

level - The priority level at which the driver for the device will execute. Must be less than or equal to the lolevel parameter of the SYS command; it may be the same as other communications devices, but must be different from the level for the COMM command, and from the levels of noncommunications tasks and devices.

channel - The channel number of the device.

label - A label, assumed to be a location definition, which must be defined in a load module. This label is the entry point of the attention subroutine. The default label for LRN 0 (operator's console) is ZIATTN, which is defined in the executive load module. The default for other LRN's is null.

modem - A number specifying the type of data set. Possible values are:

0 - direct connect

1 - Bell 1xx-type modem (103A,113F,etc). Both the data-set-ready and carrier-detect signals needed for a connection; the lack of both signals is a disconnection.

2 - Bell 2xx-type modem (201A,201C,208A,etc.) The data set ready signal needed for a connection; lack of this signal is disconnection.

3 or greater - User-defined modem type (see MODEM command).

The default is modem type 1.

speed - The data rate in bits per second. Possible values are:

50	300	2400
75	600	3600
100 (default)	900	4800
134.5	1200	7200
150	1800	9600

When this command is processed, an implicit ADMOD command is issued to include the teletype line type processor (ZQPTTY) in the load list.

VIP

VIP Command

This command identifies each visual information projection (VIP) device in the application. The format of the VIP command is:

VIPΔlrn,level,channel[,label][,modem]

VIP - Command mnemonic.

lrn - The logical resource number by which the device is requested. It must be less than or equal to the hilrn parameter of the SYS command.

level - The priority level at which the driver for the device will execute. Must be less than or equal to the lolevel parameter of the SYS command; it may be the same as other communications devices, but must be different from the level parameter in the COMM command, and from the levels specified for noncommunications tasks and devices.

channel - The channel number of the device.

label - A label, assumed to be a location definition, which must be defined in a load module. This label is the entry point of the attention subroutine. The default is null.

modem - A number specifying the type of data set. Possible values are:

0 - Direct connect

1 - Bell 1xx-type modem (103A,113F,etc.). Both data-set-ready and carrier-detect signals needed for a connection; the lack of both signals is a disconnection.

2 - Bell 2xx-type modem (201A,201C,208A,etc.). The data set ready signal needed for a connection; lack of this signal is a disconnection.

3 or greater - User-defined modem type. (See MODEM command.)

The default is modem type 2.

When this command is processed, an implicit ADMOD command is issued to include the VIP line type processor (ZQPVIP) in the load list.

*

*Command (Comments)

The comments command is used only for documenting the command input listing. These comments are bypassed by the CLM. The format of the command is shown below.

*Δcomments

* - Command mnemonic.

comments - A string of ASCII characters, up to one line in length, specifying the comments.

APPENDIX B
PLANNING AND BUILDING WITH EXECUTIVE OBJECT MODULES

The Honeywell-supplied diskette contains a map file, CLMMAP. Members of this file document the Linker command and Linker map output for the load modules created by Honeywell. This appendix indicates approaches you may use to create your own Executive and/or driver load modules from Executive object modules.

CREATING EXECUTIVE LOAD MODULES

The Honeywell Executive, ZXEX03, consists of the object modules shown in Table B-1.

Table B-1. Executive Object Modules

Description	Object Module Name
Task Manager	ZXTSKM.O
Clock Manager (basic)	ZXCMGR.O
Clock Manager (time-of-day, date)	ZXCTDA.O
Operator Interface Manager (console)	ZIOIM.O
Operator Interface Manager (panel)	ZIOIMP.O
Trace Trap Handler	ZXTRCM.O
I/O Subroutines	ZIOSUB.O
System Error Handler	ZUERR.O
Executive Initialization	ZXIN03.O
Semaphore Routine	ZXSEM.O

If you want to omit one or more of the Executive functions, you must build your own load module from the Executive object modules.

Listings of the ZXEX03 modules indicate which modules are being initialized. The general procedure for you to follow when preparing your own executive load module is to examine the existing load module's initialization code for an explanation of its functions.

The initialization subroutine table (IST) at the start of the initialization module is composed of at least one subroutine entry per module served. This means that a module being deleted should have its initialization subroutine(s) deleted. The converse is also true; a user-created module which had initialization requirements could be added to the existing IST (assuming the source was available.)

Figure B-1 shows the general structure of initialization subroutines along with a detailed sample IST.

```
ZXXIST  DC     0          START OF IST
        RESV   $AF, 0
        DC     <ZXOBJ1    OBJECT 1 INITIALIZE
        DC     0          PARAMETERS
        DC     0
        DC     <ZXOBJ2    OBJECT 2 INITIALIZE
        DC     0          PARAMETERS
        DC     0
        DC     <ZXOBJ3    OBJECT 3 INITIALIZE
        DC     0          PARAMETERS
        DC     0
        RESV   $AF, 0     END OF IST
```

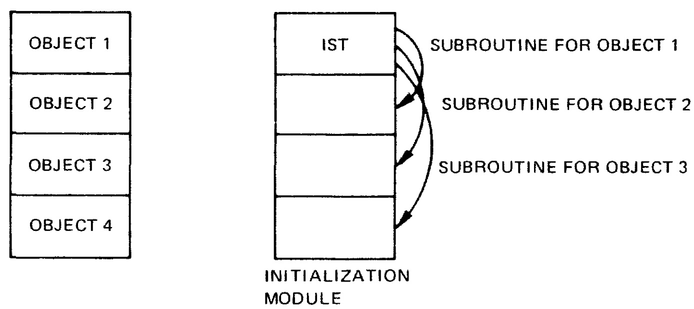


Figure B-1. Initialization Processing

Subroutines of Honeywell-supplied initialization code are functionally independent.

To summarize, there are two areas of concern when creating your own load module:

- Proper Linker commands, especially EDEF's needed by CLM.
- Proper IST subroutine entries in new initialization for object modules used by new load module.

Linker commands may be determined by examination of Honeywell-supplied link maps (CLMMAP). IST entries may be constructed by examination of Honeywell initialization module listings.

As an example of a user-created executive load module, assume that the application needs the following existing Executive functions: task, both clock functions, the operator interface for the console, the error subroutines. Furthermore, the Buffer Manager is to be added.

A listing of ZXEX03 must be examined along with the listing for the Buffer Manager (ZXBM01) for IST contents. Note that some of the Buffer Manager initialization code must be permanently resident, and cannot be overlaid. A new initialization load module is created containing all the required initialization code for the functions to be included in the new executive load module. The IST is located to accommodate the buffer initialization requirements. (See Figure B-2.)

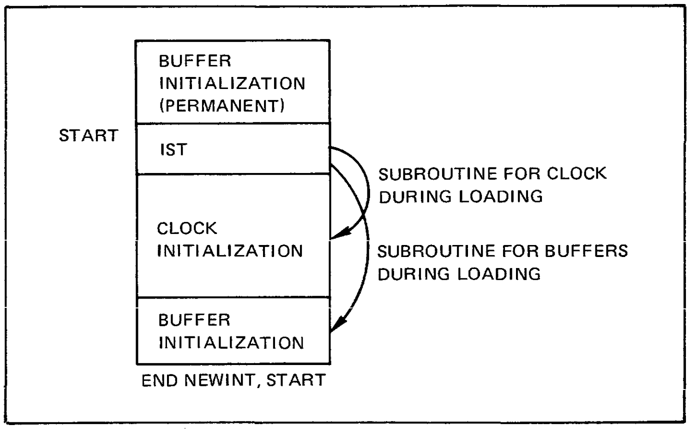


Figure B-2. New Initialization Modules

The link commands (from CLMMAP) for the new load module are all those required for the modules to be used. Note that the LINKN for the ZXEX03 initialization (ZXIN03) is not used because the entire Executive module is not being used. After all the link commands have been collected (including all necessary EDEF's), the Linker is then executed to produce the new load module that now contains the required executive functions.

I/O drivers should remain separate load modules to retain variable selection during configuration using the CLM device commands (DEVICE, TTY, VIP, BSC), and to maintain compatibility with CLM embedded file/member and start address names.

APPENDIX C
APPLICATION CONFIGURATION EXAMPLE

This appendix provides two examples of application programs. Each example contains the Linker, and CLM commands for the application along with a listing of the application program.

The first example presents an input/output program, BRDCST, whose purpose is to exercise the various device drivers provided with the BES software.

The second example is a communications test program, COM200.

CONFIGURATION COMMANDS FOR SAMPLE INPUT/OUTPUT APPLICATION

The following configuration commands are used to configure the sample application, No SYS or TSA command is used since default values are assumed.

```
CLOCK 60,50
ADMOD PROGFILE:ZXEX03,X'1200'   (Execution LM)
ADMOD USRPGS:BRDCST,X'1200'     (Application LM)
TASK BRDCST,1,10,YACT           Application LVL 10 (initially active)
DEVICE CDR,2,7,X'0580'          Card reader LVL 7
DEVICE LPT,5,9,X'1300'          Line printer LVL 9
DEVICE DSK,4,8,X'1200'          Disk LVL 8
DEVICE ASR,0,6,X'1380'          Operator's console LVL 6
EQLRN 3,6                       Teletype LVL 6
ATLRN 6,11                      Input Task LVL 11
ATLRN 7,12                      Output Task LVL 12
ATLRN 8,8                       Disk out LVL 8
EQLRN 9,6                       Teletype out LVL 6
EQLRN 10,6                      ASR in LVL 6
EQLRN 11,6                      ASR out LVL 6
ATLRN 12,13                     Output Task 2 LVL 13
ATLRN 13,14                     Output Task 3 LVL 14
QUIT HLT
```

LINK COMMANDS FOR SAMPLE INPUT/OUTPUT PROGRAM

The following commands are used to produce the load module BRDCST prior to invoking CLM to configure an application using the program:

```
NAME BRDCST
LINK BRDCST
EDEF BRDCST
MAP
QUIT
```

SAMPLE INPUT/OUTPUT PROGRAM

The following pages show a documented listing of the BRDCST program.

```
                              TITLE BRDCST
                   *
                   *          THIS TEST PROGRAM IS A
                   *          MEDIA TRANSCRIPTION TEST.
                   *          IT CAN EXECUTE AS AN
                   *          ON-LINE OR OFF-LINE
                   *          DRIVER TEST......
                   *
                   *          THE OPERATOR WILL TYPE
                   *          0X0Y0Y0Y
                   *
                   *           X= INPUT DEV NUMBER Y= OUTPUT DEV NUMBER
                   *           0= CARD READER
                   *           1= TELETYPE
                   *           2= DISKETTE IN
                   *           3= PRINTER
                   *           4= DISKETTE OUT
                   *           5= TELETYPE OUT
                   *           6= ASR IN
                   *           7= ASR OUT
                   *
                   *
                   *          LRN   0=OP CONSOLE
                   *          LRN   1=CONTROL TASK
                   *          LRN   2=CARD READER
                   *          LRN   3=TELETYPE IN
                   *          LRN   4=DISKETTE IN
                   *          LRN   5=PRINTER
                   *          LRN   6=INPUT TASK
                   *          LRN   7=OUTPUT TASK
                   *          LRN   8=DISKETTE OUT
                   *          LRN   9=TELETYPE OUT
                   *          LRN   10=ASR IN
                   *          LRN   11=ASR OUT
                   *          LRN   12=OUTPUT TASK2
                   *          LRN   13=OUTPUT TASK3
                   *
                   *
                   ******
                   *****
                   ****       LRN TABLE
                   ***
                   **
                   *
                   *
0000  2020         BLNKS      TEXT  '  '
0001  3939 3939    TERM       TEXT  '9999'          TERMINATION CHARACTERS
                   *
                   *
0003               DEVTBL     RESV  0
0003  000B                    DC    <CRDBLK         LRN 2
0004  0013                    DC    <TTYBLK         LRN 3
0005  001B                    DC    <DSKBLI         LRN 4
0006  002B                    DC    <PRTBLK         LRN 5
0007  0023                    DC    <DSKBLO         LRN 8
0008  0033                    DC    <TTYOUT         LRN 9
0009  003B                    DC    <ASRINP         LRN 10
000A  0043                    DC    <ASROUT         LRN 11
                   *
```

```
        0018        DISCOM   EQU      $
0018    0000                 RESV     $AF,0
0019    0001                 DC       X'01'          WAIT TILL I/O COMPLETE
001A    0408                 DC       X'408'         DISCONNECT
001B    0000                 RESV     $AF,0          BUFFER
001C    0001                 DC       1              RANGE=1 WORD
001D    0030                 DC       X'30'
001E    0000                 DC       X'0'
001F    0000                 DC       X'0'
                    *                 *
                    *        *        *IORB:COMMUNICATIONS CONSOLE INPUT*
                    *                 *
        0020        COMCI    EQU      $
0020    0000                 RESV     $AF,0
0021    0001                 DC       X'01'          WAIT TILL I/O COMPLETE
0022    0402                 DC       X'402'         READ
0023    01F8                 DC       <COMBFR        BUFFER ADDRESS
0024    0048                 DC       72             RANGE=72
0025    0030                 DC       X'30'          ECHO, L.F. &C/R
0026    0000                 DC       X'0'
0027    0000                 DC       X'0'
                    *                 *
                    *        *        *IORB:COMMUNICATIONS CONSOLE OUTPUT
                    *                 *
        0028        COMCO    EQU      $
0028    0000                 RESV     $AF,0
0029    0001                 DC       X'01'          WAIT TILL I/O COMPLETE
002A    0441                 DC       X'441'         WRITE, CONTROL BYTE RIGHTMOST
002B    01A6                 DC       <LPBUF1        BUFFER START ADDRESS
002C    0049                 DC       73             73=RANGE
002D    0030                 DC       X'30'          ECHO, LINE FEED  AND C/R
002E    0000                 DC       X'0'           RESIDUAL RANGE
002F    0000                 DC       X'0'           STATUS WORD
                    *                 *
                    *        *        *IORB:CARD READER INPUT
                    *                 *
        0030        CDRIN    EQU      $
0030    0000                 RESV     $AF,0
0031    0001                 DC       X'01'          WAIT
0032    0302                 DC       X'302'         READ CARDS
0033    01CF                 DC       <CDRBUF        BUFFER
0034    0050                 DC       80             80-CHARACTER RANGE
0035    0000                 DC       X'0'           ASCII MODE
0036    0000                 DC       X'0'
0037    0000                 DC       X'0'
                    *                 *
                    *        *        *IORB:LINE PRINTER OUTPUT
                    *                 *
        0038        LPTOUT   EQU      $
0038    0000                 RESV     $AF,0
0039    0001                 DC       X'01'          WAIT
003A    0241                 DC       X'241'         WRITE/CONTROL BYTE RIGHTMOST
003B    01A6                 DC       <LPBUF1        BUFFER
003C    0049                 DC       73
003D    0000                 DC       X'0'
003E    0000                 DC       X'0'
003F    0000                 DC       X'0'
                    *                 *
                    *        *        *BEGIN
                    *                 *
```

```
                    *
                    *               TASK 1 (INPUT) REQUEST BLOCK LRN 6
                    *
009C  0000          TASK01    RESV  1,0
009D  0000                    DC    X'0000'
009E  0600                    DC    X'0600'
009F  013F                    DC    <INSTRT
                    *
                    *               TASK 2 (OUTPUT) REQUEST BLOCK LRN 7
                    *
00A0  0000          TASK02    RESV  1,0
00A1  0000                    DC    X'0000'
00A2  0700                    DC    X'0700'
00A3  0150                    DC    <OUTSTR
                    *
                    *****
                    **        OUTPUT2 TASK REQUEST BLOCK
                    *****
                    *
00A4  0000          TASK03    RESV  1,0
00A5  0000                    DC    X'0000'
00A6  0C00                    DC    X'0C00'
00A7  0161                    DC    <OUT2ST
                    *
                    *****
                    **        OUTPUT3 TASK REQUEST BLOCK
                    *****
                    *
00A8  0000          TASKC4    RESV  1,0
00A9  0000                    DC    X'0000'
00AA  0D00                    DC    X'0D00'
00AB  0171                    DC    <OUT3ST
                    *
                    *
                    *
                    *
                    *               START-UP I/O REQUEST BLOCK(OUT)
                    *
00AC  0000          MESSAG    RESV  1,0
00AD  0001                    DC    X'01'
00AE  0001                    DC    X'0001'           LRN 0
00AF  00B4                    DC    <OUTMSG
00B0  0065                    DC    101
00B1  0000                    RESV  3,0
00B4  2063 7264 2072 OUTMSG   TEXT  ' crd rdr=0'
00B7  6472 3D30
00B9  2074 7479 2069          TEXT  ' tty in=1'
00BC  6E3D 3100
00BE  2064 6973 6B20          TEXT  ' disk in=2'
00C1  696E 3D32
00C3  2070 726E 7472          TEXT  ' prntr=3'
00C6  3D33
00C7  2064 6973 6B20          TEXT  ' disk out=4'
00CA  6F75 743D 3400
00CD  0D0A                    DC    Z'0D0A'
00CE  2074 7479 206F          TEXT  ' tty out=5'
00D1  7574 3D35
00D3  2061 7372 2069          TEXT  ' asr in=6'
00D6  6E3D 3600
00D8  2061 7372 206F          TEXT  ' asr out=7'
```

```
        00DB   7574 3D37
000173  00DD   0D0A                                   DC     Z'0D0A'
000174  00DE   2074 7970 6520                         TEXT   ' type 0n0y0y0y'
        00E1   306E 3079 3079
               3079
000175                                 *
000176                                 *              STARTUP I/O REQUEST BLOCK
000177                                 *
000178  00E5   0000                    INPMSG         RESV   1,0
000179  00E6   0001                                   DC     X'01'
000180  00E7   0002                                   DC     X'0002'
000181  00E8   00ED                                   DC     <INMSG
000182  00E9   0008                                   DC     8
000183  00EA   0000                                   RESV   3,0
000184  00ED   2020 2020 2020          INMSG          TEXT   '
        00F0   2020
000185                                 *
000186                                 *
000187  00F1   00AC                    PARLST         DC     <MESSAG           PARAMETER LIST
000188  00F2   00E5                                   DC     <INPMSG
000189  00F3   0000                                   DC     0
000190  00F4   0000                                   DC     0                 WORK WORDS(PREVICUS WORD TOO)
000191                                 *
000192                                 *
000193                                 *
000194                                 *
000195                                 *
000196                                 *              I/O ERROR MESSAGE IORB
000197                                 *
000198                                 *
000199  00F5   0000                    ERRMSG         RESV   1,0
000200  00F6   0001                                   DC     X'01'
000201  00F7   0041                                   DC     X'C041'
000202  00F8   00FD                                   DC     <ERROR
000203  00F9   0011                                   DC     17
000204  00FA   0000                                   RESV   3,0
000205  00FD   0041                    ERROR          DC     X'C041'
000206  00FE   6465 763D                              TEXT   'dev='
000207  0100   0000                    DEVASN         DC     X'0000'
000208  0101   2065 7272 6F72                         TEXT   ' error='
        0104   3D00
000209  0105   0000                    STATUS         DC     X'0000'
000210                                 *
000211                                 *
000212         0001                    $LVCT1         EQU    1
000213         0002                    $LVCT2         EQU    2
000214         0005                    $LVCT5         EQU    5
000215         0040                    $LWBT          EQU    X'40'
000216         00EF                    OUTSK2         EQU    <INMSG+2
000217         00F0                    OUTSK3         EQU    <INMSG+3
000218                                 *
000219                                 *
000220                                 *
000221                                 *              TYPE STARTUP MSG,WAIT FOR REPLY
000222                                 *
000223  0106   CC80 00F2               BRDCST         LDB    $B4,<PARLST+$LVCT1
000224  0108   9870 2020                              LDR    $R1,=Z'2020'
000225  010A   9F00 00EE                              STR    $R1,<INMSG+1
000226  010C   9F00 00EF                              STR    $R1,<INMSG+2
000227  010E   9F00 00F0                              STR    $R1,<INMSG+3
```

```
0181  82C4 0001        IOK       LB    $B4.$LVCT1,X'0020' DO BIT TEST
0183  0020
0184  0580 018B                  BBF   <NOTDSK            NOT A DISK IF BIT=0
0186  B844 0005                  LDR   $R3,$B4.$LVCT5     LOAD CURRENT SECTOR INTO R3
0188  8AD3                       INC   =$R3               BUMP IT BY 1
0189  BF44 0005                  STR   $R3,$B4.$LVCT5     PUT IT BACK WHERE IT BELONGS...
018B  D380 0000   X    NOTDSK    LNJ   $B5,<ZXTERM        NOW YOU CAN POST + TERMINATE
                       *
                       *         THE FOLLOWING CODE IS AN ERROR ROUTINE
                       *         IT DISPLAYS THE DEVICE NUMBER AND THE
                       *         STATUS CODE ON THE OP CONSOLE THEN IT
                       *         RETURNS CONTROL AT THE POINT THE I/O
                       *         ORDER WAS REQUESTED. IT RUNS AT THE
                       *         LEVEL OF THE OFFENDING ROUTINE......
                       *
018D  8951             ERROLT    LBT   =$R1,Z'3030'       MAKE IT TYPEABLE
018E  3030
018F  9F00 0105                  STR   $R1,<STATUS         THROW STATUS IN BUFFER
0191  AF00 0100                  STR   $R2,<DEVASN        THROW DEVICE NUMBER IN SAME
0193  CF80 019C                  STB   $B4,<STORB4        HOLD RETURN ADDRESS
0195  CB80 00F5                  LAB   $B4,<ERRMSG        AND SETUP FOR
0197  D380 0000   X              LNJ   $B5,<ZIOREQ        THE ERROR TYPEOUT
0199  CC80 019C                  LDB   $B4,<STORB4        LOAD RETURN ADDRESS IN B4
019B  8384                       JMP   $B4                GO BACK AND RE-ISSUE ORDER
                       *
019C  0000             STORB4    RESV  1,0                B4 HOLD WORD
                       *
                       *
                       *
                       *         XDEFS AND XLOCS
                       *
      0150                       XDEF  OUTSTR,INSTRT,BRDCST
      013F
      0106
                                 XLOC  ZITYPR,ZXRQST,ZIOREQ,ZXTERM
019D                             END   BRDCST
0000  ERR COUNT
```

CONFIGURATION COMMANDS FOR SAMPLE COMMUNICATIONS APPLICATION

The following commands are used to configure the sample communications application. The absence of a TSA command indicates that default values were taken.

```
LACT CLMCOMM:COMM
ELACT
SYS 16,22,SAF
OIM 5,11
COMM 7
TTY 4,10,X'FD00',,0
ADMOD PROGFILE:ZXEX03,X'0400'
ADMOD LINKFILE:COM200,X'0480'
CLOCK ,50
DATE '760622','1200'
DEVICE KSR,0,6,X'0500'
DEVICE LPT,2,8,X'1380'
DEVICE CDR,3,9,X'1300'
TASK GLUE,6,12,YACT
QUIT HLT,MAP
```

LINK COMMANDS FOR SAMPLE COMMUNICATIONS PROGRAM

The following commands are used to produce the load module COM200 prior to invoking CLM to configure an application using this program:

```
NAME COM200
LINK COM200
MAP
EDEF GLUE
QUIT
```

SAMPLE COMMUNICATIONS PROGRAM

The following pages show a documented listing of the COM200 pagram.

```
**   COM200     LINK MAP
**START
**LOW       0000
**HIGH      0246
**CURRENT 0246

**EXT DEFS
P    ZHCOMM   0000
P    ZHREL    0000
 *   COM200   0000
     GLUE     0040

**UNDEF
 *   COM200   0000
     ZIOREQ   0191
     ZXCTOD   011E
     ZXCMGR   0127

IMG UNDEF
```

```
                                  TITLE   COM200,'760622' COMM TEST PROGRAM
                          *               *
                          *               *EXTERNALS
                          *               *
                                  XLOC    ZIOREQ              INPUT-OUTPUT DRIVER
                                  XLOC    ZXCTOD
                                  XLOC    ZXCMGR              CLOCK MANAGER
                                  XLOC
                                  XDEF    GLUE
           0040
                          *               *
                          *       *       *IORB:LABELED DISPLACEMENTS
                          *               *
           0000           ZIRLNK  EQU     0                   LINK
           0001           ZIRCT1  EQU     ZIRLNK+$AF          CONTROL WORD 1
           0002           ZIRCT2  EQU     ZIRCT1+1            CONTROL WORD 2
           0003           ZIRBAD  EQU     ZIRCT2+1            BUFFER ADDRESS
           0004           ZIRRNG  EQU     ZIRBAD+$AF          RANGE
           0005           ZIRDVS  EQU     ZIRRNG+1            DEVICE SPECIFIC
           0006           ZIRRSR  EQU     ZIRDVS+1            RESIDUAL RANGE
           0007           ZIRST1  EQU     ZIRRSR+1            SOFTWARE STATUS WORD
                          *               *
                          *       *       *IORB:CONSOLE INPUT*                  *
                          *               *
           0000           KSRIN   EQU     $
     0000  0000                   RESV    $AF,0
     0001  0001                   DC      X'01'               WAIT TILL I/O COMPLETE
     0002  0002                   DC      X'02'               READ
     0003  01A8                   DC      <KSRBUF             BUFFER ADDRESS
     0004  0006                   DC      6                   RANGE IN BYTES
     0005  0030                   DC      X'30'               C/R, LINE FEED, &ECHO
     0006  0000                   DC      X'0'                RESIDUAL RANGE
     0007  0000                   DC      X'0'                STATUS
                          *               *
                          *       *       *IORB:CONSOLE OUTPUT
                          *               *
           0008           KSROUT  EQU     $
     0008  0000                   RESV    $AF,0
     0009  0001                   DC      X'01'               WAIT
     000A  0041                   DC      X'41'               WRITE
     000B  01AC                   DC      <LPBUF1             BUFFER
     000C  0014                   DC      20                  RANGE IN BYTES
     000D  0030                   DC      X'30'               C/R, L.F., & ECHO
     000E  0000                   DC      X'0'
     000F  0000                   DC      X'0'
                          *               *
                          *       *       *IORB:CONNECT COMMUNICATIONS CONSOLE*
                          *               *
           0010           CONCOM  EQU     $
     0010  0000                   RESV    $AF,0
     0011  0001                   DC      X'01'               WAIT TILL I/O COMPLETE
     0012  040A                   DC      X'40A'              CONNECT
     0013  0000                   RESV    $AF,0
     0014  0001                   DC      1                   RANGE=1 WORD
     0015  0030                   DC      X'30'
     0016  0000                   DC      X'0'
     0017  0000                   DC      X'0'
                          *               *
                          *       *       *IORB:DISCONNECT COMMUNICATIONS CONSOLE*
                          *               *
```

```
0008  0000          CRDBLK     RESV  1,0            CARD READER
000C  0001                     DC    X'01'          I/O
000D  0202                     DC    X'0202'        CONTROL
000E  004C                     DC    <BUFFER        BLOCK
000F  0050                     DC    80
0010  0000                     RESV  3,0
                    *
0013  0000          TTYBLK     RESV  1,0            TTY(INPUT)
0014  0001                     DC    X'01'          I/O
0015  0342                     DC    X'0342'         CONTROL
0016  004B                     DC    <BUFFEP         BLOCK
0017  0050                     DC    80
0018  0000                     RESV  3,0
                    *
001B  0000          DSKBLI     RESV  1,0            DISKETTE INPUT
001C  0021                     DC    X'0021'        I/O
001D  0402                     DC    X'0402'         CONTROL
001E  004C                     DC    <BUFFER        BLOCK
001F  0050                     DC    80
0020  0000                     RESV  3,0
                    *
0023  0000          DSKBLO     RESV  1,0            DISKETTE OUTPUT
0024  0021                     DC    X'0021'        I/O
0025  0801                     DC    X'0801'         CONTROL
0026  004C                     DC    <BUFFER        BLOCK
0027  0050                     DC    80
0028  0000                     RESV  3,0
                    *
002B  0000          PRTBLK     RESV  1,0            PRINTER
002C  0001                     DC    X'01'          I/O
002D  0541                     DC    X'0541'        CONTROL
002E  004B                     DC    <BUFFEP        BLOCK
002F  0050                     DC    80
0030  0000                     RESV  3,0
                    *
0033  0000          TTYOUT     RESV  1,0            TTY (OUTPUT)
0034  0001                     DC    X'01'          I/O
0035  0941                     DC    X'0941'        CONTROL
0036  004B                     DC    <BUFFEP        BLOCK
0037  0050                     DC    80
0038  0000                     RESV  3,0
                    *
003B  0000          ASRINP     RESV  1,0            ASR (INPUT)
003C  0001                     DC    X'01'          I/O
003D  0A02                     DC    X'0A02'        CONTROL
003E  004C                     DC    <BUFFER        BLOCK
003F  0050                     DC    80
0040  0044                     DC    X'0044'
0041  0000                     RESV  2,0
                    *
0043  0000          ASROUT     RESV  1,0            ASR (OUTPUT)
0044  0001                     DC    X'01'          I/O
0045  0B01                     DC    X'0B01'        CONTROL
0046  004C                     DC    <BUFFER        BLOCK
0047  0050                     DC    80
0048  0044                     DC    X'0044'
0049  0000                     RESV  2,0
                    *
004B  0041          BUFFEP     DC    X'0041'        CARRIAGE CONTROL CHARACTER PRINT + SPACE
004C                BUFFER     RESV  80
```

```
0040  CB80 0010       GLUE     LAB    $R4,<CONCOM         GET CONNECT IORB
0042  D380 0000   X            LNJ    $R5,<ZIOREQ         AND CONNECT COMMUNICATIONS CONSOLE
0044  F380 0067                LNJ    $R7,<CLEARR         CLEAR MESSAGE BUFFER
0046  B820 0236       QUERY    LDR    $R3,<QUFS,$R2       GET MESSAGE TEXT
0048  BF20 01A7                STR    $R3,<LPBUF2,$R2     AND STORE ADDRESS
004A  2E01                     ADV    $R2,1               ADD 1 TO COUNTER
004B  2D04                     CMV    $R2,4               8 CHARACTERS?
004C  0981 FFF9                BNE    QUERY               IF NOT, GET MORE
004E  B800 021C                LDR    $R3,<BELLS          GET BELLS
0050  BF20 01A7                STR    $R3,<LPBUF2,$R2     AND STORE IN BUFFER
0052  CB80 0008                LAB    $R4,<KSROUT
0054  4C09                     LDV    $R4,9
0055  CF44 0004                STR    $R4,$R4.ZIRRNG
0057  D380 0000   X            LNJ    $R5,<ZIOREQ         PRINT MESSAGE ON KSR
0059  0F81 0007                B      KREAD               AND THEN GO TO READ KS
005B  CB80 0018       DISCON   LAB    $R4,<DISCOM         GET DISCONNECT IORB
005D  D380 0000   X            LNJ    $R5,<ZIOREQ         DISCONNECT COMM CONSOLE
005F  0F81 01E3                B      FINIS
0061  CB80 0000       KREAD    LAB    $R4,<KSRIN          LOAD KSR INPUT IORB
0063  D380 0000   X            LNJ    $R5,<ZIOREQ                   READ COMMAND
0065  0F81 0050                B      GETCH
                      *               *
                      *        *      *ERROR ROUTINES
                      *               *
0067  CB80 0008       CLEARR   LAB    $R4,<KSROUT         GET IORB
0069  AB80 01A6                LAB    $R2,<LPBUF1         GET CONTROL BYTE
006B  AFC4 0003                STB    $R2,$R4.ZIRRAD      AND STORE ADDRESS
006D  8752                     CL     =$R2                CLEAR COUNTER
006E  BB80 01A7                LAB    $R3,<LPBUF2         GET LPBUF2
0070  2C24                     LDV    $R2,36              SET COUNTER TO 36
0071  D380 014E                LNJ    $R5,<SPACIT         GO CLEAR BUFFER
0073  8752                     CL     =$R2                CLEAR COUNTER AGAIN
0074  8387                     JMP    $R7                 AND GO BACK WHERE YOU CAME FROM
                      *               *
0075  4C0F            ERRMSG   LDV    $R4,15
0076  CF44 0004                STR    $R4,$R4.ZIRRNG
0078  B820 021D                LDR    $R3,<MSG1,$R2       GET MESSAGE TEXT
007A  BF20 01A7                STR    $R3,<LPBUF2,$R2     AND STORE ADDRESS
007C  2E01                     ADV    $R2,1               ADD 1 TO COUNTER
007D  2D07                     CMV    $R2,7               14 CHARACTERS?
007E  0981 FFF6                BNE    ERRMSG              IF NOT, GET MORE TEXT
0080  D380 0000   X            LNJ    $R5,<ZIOREQ         ELSE,GO PRINT IT
0082  F380 0067                LNJ    $R7,<CLEARR         CLEAR BUFFER AGAIN
0084  8752                     CL     =$R2
0085  0F81 FFC0                B      QUERY
                      *               *
0087  4C0B            DONMES   LDV    $R4,11
0088  CF44 0004                STR    $R4,$R4.ZIRRNG
008A  B820 0224                LDR    $R3,<MSG2,$R2       GET MESSAGE TEXT
008C  BF20 01A7                STR    $R3,<LPBUF2,$R2   AND STORE ADDRESS
008E  2E01                     ADV    $R2,1               ADD 1 TO COUNTER
008F  2D04                     CMV    $R2,4               8 CHARACTERS?
0090  0981 FFF6                BNE    DONMES              IF NOT, GET MORE TEXT
0092  B800 021C                LDR    $R3,<BELLS
0094  BF20 01A7                STR    $R3,<LPBUF2,$R2
0096  D380 0000   X            LNJ    $R5,<ZIOREQ         GO PRINT MESSAGE
0098  0F81 FFC2                B      DISCON              AND THEN GO DISCONNECT COMM CONSOLE
                      *               *
009A  4C0D            DEVERR   LDV    $R4,13
009B  CF44 0004                STR    $R4,$R4.ZIRRNG
```

```
009D  8820 0228                LDR    $R3,<MSG3,$R2
009F  BF20 01A7                STR    $R3,<LPBUF2,$R2       STORE ADDRESS OF MESSAGE
00A1  2E01                     ADV    $R2,1                 ADD 1 TO COUNTER
00A2  2D06                     CMV    $R2,6                 12 CHARACTERS?
00A3  0981 FFF6                BNE    DEVERR    IF NOT GET MORE TEXT
00A5  D380 0000    X           LNJ    $B5,<ZIOREQ           OTHERWISE, GO PRINT IT
00A7  8752                     CL     =$R2
00A8  0F81 FF9D                B      QUERY
                         *            *
00AA  F380 0067          TWIG1 LNJ    $B7,<CLEARR
00AC  0F81 FFDA                B      DONMES
00AE  F380 0067          TWIG2 LNJ    $B7,<CLEARR
00B0  0F81 FFC4                B      ERRMSG
00B2  F380 0067          TWIG3 LNJ    $B7,<CLEARR
00B4  0F81 FFE5                B      DEVERR
                         *            *
                         *     *      *READ CONSOLE INPUT*              *
                         *            *
00B6  8800 01A2          GETCH LDR    $R3,<KSRBUF           GET 1ST TWO CHARACTERS
00B8  8900 01F7                CMR    $R3,<TERM
00BA  0901 FFEF                BE     TWIG1
00BC  8900 023D                CMR    $R3,<CH2              COMPARE TO CA
00BE  0981 FFEF                BNE    TWIG2                 WRONG COMMAND
00C0  8800 01A3                LDR    $R3,<KSRBUF+1         GET 2ND TWO CHARACTERS
00C2  8900 023E                CMR    $R3,<CH4              COMPARE TO RD
00C4  0981 FFE9                BNE    TWIG2                 WRONG COMMAND
00C6  8800 01A4                LDR    $R3,<KSRBUF+2         GET NEXT TWO CHARACTERS
00C8  8900 023F                CMR    $R3,<CH6              COMPARE TO IN
00CA  0981 FFE3                BNE    TWIG2                 WRONG COMMAND
00CC  0F81 0001                B      CARDRD
                         *            *
                         *     *      *READ CARDS
                         *            *
00CE  C880 0030          CARDRD LAB   $B4,<CDRIN            GET IORB
00D0  D380 0000    X           LNJ    $B5,<ZIOREQ           READ A CARD
00D2  8800 01CF                LDR    $R3,<CDRBUF           LOAD IT IN
00D4  8970 1D20                CMR    $R3,=Z'1D20'                    COMPARE TO EOF
00D6  0901 0090                BE     COMQU1                IF EOF, GO TO COMM CONSOLE
00D8  8752                     CL     =$R2
00D9  8754                     CL     =$R4
00DA  1901 0003                BEZ    $R1,CHECK             IF NO ERROR, GO TO CHECK
00DC  0F81 FFD5                B      TWIG3                 OTHERWISE, DEVICE ERROR
00DE  B2A0 01CF          CHECK LLH    $R3,<CDRBUF,$R2
00E0  2E01                     ADV    $R2,1
00E1  B1F0 2000                CMH    $R3,=Z'20'
00E3  0981 0006                BNE    COUNT
00E5  2D50                     CMV    $R2,80
00E6  0901 0009                BE     COMOUT
00E8  0F81 FFF5                B      CHECK
00EA  C852               COUNT LDR    $R4,=$R2
00EB  2D50                     CMV    $R2,80
00EC  0901 0003                BE     COMOUT
00EE  0F81 FFEF                B      CHECK
                         *            *
                         *     *      *SEND CARD INPUT TO COMMUNICATIONS DEVICE*
                         *            *
00F0  C880 0028          COMOUT LAB   $B4,<COMCO            GET IORB
00F2  AB80 01A6                LAB    $B2,<LPBUF1           GET CONTROL BYTE
00F4  AFC4 0003                STB    $B2,$B4.ZIRBAD        AND STORE ADDRESS
00F6  4E01                     ADV    $R4,1
```

```
00F7  CF44 0004                        STR    $R4,$B4.ZIRRNG
00F9  8752                             CL     =$R2                 CLEAR R2 COUNTER TO ZERO
00FA  B820 01CF           COMMO        LDR    $R3,<CDRBUF.$R2      GET TWO CHARACTERS FROM CDRBUF
00FC  BF20 01A7                        STR    $R3,<LPBUF2.$R2      AND STORE IN LPBUF2
00FE  2E01                             ADV    $R2,1                ADD 1 TO COUNTER
00FF  2D24                             CMV    $R2,36               COMPARE TO 36
0100  0981 FFF9                        BNE    COMMO                IF NOT 36,GET MORE
0102  D380 0000      X                 LNJ    $B5,<ZIOREQ          OTHERWISE SEND TO COMM DEVICE
0104  1981 FFAD                        BNEZ   $R1,TWIG3
0106  F3C0 0003                        LNJ    $B7,GETIME
0108  0F81 FFC5                        B      CARDRD
                          *                   *
                          *            *      *GET DATE AND TIME AND PRINT IT*
                          *                   *
010A  CB80 0000      X    GETIME       LAB    $B4,<ZXCTOD          GET DATE AND TIME
010C  8752                             CL     =$R2
010D  BB80 022E                        LAB    $B3,<TIMER
010F  2C08                             LDV    $R2,8
0110  D380 014E                        LNJ    $B5,<SPACIT
0112  8F84                             RSTR   $B4,=Z'1F00'         RESTORE IN R-REGISTERS 3-7
0113  1F00
0114  8F40 0119                        SAVE   TIMER,=Z'1F00'       AND SAVE IN TIMER
0116  1F00
0117  1C04                             LDV    $R1,=4               PUT 4 IN R1
0118  CB80 022E                        LAB    $B4,<TIMER           GET ADDR OF DATE
011A  D380 0000      X                 LNJ    $B5,<ZXCMGR          CONVERT DATE TO ASCII
011C  1C03                             LDV    $R1,=3               PUT 3 IN R1
011D  CB80 0000      X                 LAB    $B4,<ZXCTOD
011F  8F84                             RSTR   $B4,=Z'1F00'
0120  1F00
0121  8F40 010F                        SAVE   TIMER+3,=Z'1F00'
0123  1F00
0124  CB80 0231                        LAB    $B4,<TIMER+3         GET ADDR OF TIME
0126  D380 0000      X                 LNJ    $B5,<ZXCMGR          CONVERT TIME TO ASCII
                          *                   *
0128  CB80 0038           PRTM1        LAB    $B4,<LPTOUT          GET IORB
012A  AB80 01A6                        LAB    $B2,<LPBUF1          GET CONTROL BYTE
012C  AFC4 0003                        STB    $B2,$B4.ZIRBAD       STORE ADDR
                          *                   *
012E  8752                ZAPIT        CL     =$R2                 SET R2 TO 0
012F  BB80 01A7                        LAB    $B3,<LPBUF2          GET LPT BUFFER
0131  2C24                             LDV    $R2,=36              PUT 36 IN R2
0132  D380 014E                        LNJ    $B5,<SPACIT          CLEAR BUFFER TO SPACES
                          *                   *
0134  C840 010C           PRTM2        LDR    $R4,SLASH            GET A SLASH
0136  E854                             LDR    $R6,=$R4             ONE FOR R6 TOO
0137  B800 022E                        LDR    $R3,<TIMER           LOAD YEAR IN R3
0139  8FC0 00F5                        RSTR   TIMER+1,=Z'0500'     LOAD REST OF DATE IN R5 AND R7
013B  0500
013C  8F40 006A                        SAVE   LPBUF2,=Z'1F00'      AND SAVE R3-R7 INTO LPBUF2
013E  1F00
013F  C840 0102                        LDR    $R4,COLON            PUT A COLON IN R4
0141  E854                             LDR    $R6,=$R4             AND ALSO IN R6
0142  8FC0 00FE                        RSTR   TIMER+3,=Z'7500'     GET TIME AND PUT IN  OTHER R-REGISTERS
0144  7500
0145  8F40 0067                        SAVE   LPBUF2+6,=Z'7F00'    AND SAVE INTO LPBUF2
0147  7F00
0148  D380 0000      X                 LNJ    $B5,<ZIOREQ           GO PRINT IT
014A  1901 0008                        BEZ    $R1,PRINT            IF STATUS OK, GO ON
014C  0F81 FF65                        B      TWIG3                OTHERWISE, DEVICE ERROR
```

```
                               *                  *
          014E                 SPACIT    EQU      $
    014E  F800 0240                      LDR      $R6,<BLANKS
    0150  EF6B                 $A        STR      $R6,$B3.-$R2
    0151  29FF           T               BNEZ     $R2,>-$A
    0152  8385                           JMP      $B5
                               *                  *
                               *         *        *PRINT CARD INPUT
                               *                  *
    0153  C880 0038            PRINT     LAB      $B4,<LPTOUT         GET IORB
    0155  A880 01A6                      LAB      $B2,<LPBUF1                   GET CONTROL BYTE VALUE
    0157  AFC4 0003                      STB      $B2,$B4.ZIRBAD      PLACE IT IN BUFFER
    0159  8752                           CL       =$R2
    015A  B820 01CF            PRINT1    LDR      $R3,<CDRBUF.$R2     GET 2 CHARACS FROM CDRBUF
    015C  BF20 01A7                      STR      $R3,<LPBUF2.$R2     STORE IN LPBUF2
    015E  2E01                           ADV      $R2,1               ADD 1 TO COUNTER
    015F  2D24                           CMV      $R2,36              72 CHARACTERS?
    0160  0981 FFF9                      BNE      PRINT1              IF NOT, GET MORE
    0162  D380 0000      X               LNJ      $B5,<ZIOREQ         PRINT A LINE
    0164  1981 FF4D                      BNEZ     $R1,TWIG3
    0166  8387                           JMP      $B7
                               *                  *
                               *         *        *COMMUNICATIONS CONSOLE I/O*
                               *                  *
    0167  B880 01A7            COMQU1    LAB      $B3,<LPBUF2
    0169  8752                           CL       =$R2
    016A  2C24                           LDV      $R2,36
    016B  D380 014E                      LNJ      $B5,<SPACIT         CLEAR BUFFER TO SPACES
    016D  C880 0028                      LAB      $B4,<COMCO          GET IORB
    016F  A880 01A6                      LAB      $B2,<LPBUF1         GET CONTROL BYTE
    0171  AFC4 0003                      STB      $B2,$B4.ZIRBAD      AND STORE ADDRESS
    0173  4C07                           LDV      $R4,7
    0174  CF44 0004                      STR      $R4,$B4.ZIRRNG
    0176  8752                           CL       =$R2                CLEAR COUNTER
    0177  B820 023A            COMQUE    LDR      $R3,<QUES2.$R2
    0179  BF20 01A7                      STR      $R3,<LPBUF2.$R2     STORE MESSAGE IN OUTPUT BUFFER
    017B  2E01                           ADV      $R2,1               ADD 1 TO COUNTER
    017C  2D03                           CMV      $R2,3               6 CHARACTERS?
    017D  0981 FFF9                      BNE      COMQUE              IF NOT, READ SOME MORE
    017F  C880 0028                      LAB      $B4,<COMCO
    0181  D380 0000      X               LNJ      $B5,<ZIOREQ         OTHERWISE, SEND MESSAGE
    0183  1901 000A                      BEZ      $R1,COMIN           IF STATUS OK, GO TO READ INPUT
    0185  0F81 FF2C                      B        TWIG3               ELSE, DEVICE ERROR
    0187  B900 01F7            ENDIT     CMR      $R3,<TERM
    0189  0981 0009                      BNE      COMMI
    018B  8752                           CL       =$R2
    018C  0F81 FEB9                      B        QUERY
    018E  C880 0020            COMIN     LAB      $B4,<COMCI          GET IORB
    0190  D380 0000      X               LNJ      $B5,<ZIOREQ         READ COMM CONSOLE INPUT
    0192  8752                           CL       =$R2                CLEAR COUNTER TO ZERO
    0193  B820 01F8            COMMI     LDR      $R3,<COMBFR.$R2     GET TWO CHARACTERS
    0195  BF20 01CF                      STR      $R3,<CDRBUF.$R2     AND STORE IN CDRBUF
    0197  2E01                           ADV      $R2,1               ADD 1 TO COUNTER
    0198  2D01                           CMV      $R2,1     COMPARE TO 1
    0199  0901 FFED                      BE       ENDIT               GO CHECK IF THEY ARE TC
    019B  2D24                           CMV      $R2,36              72 CHARACTERS?
    019C  0981 FFF6                      BNE      COMMI               IF NOT, GET MORE
    019E  F3C0 FF6B                      LNJ      $B7,GETIME          ELSE,GO GET THE TIME
    01A0  0F81 FFED                      B        COMIN               GO READ MORE FROM COMM CONSOLE
                               *                  *
```

```
    000352                             *            *           *DEFINITIONS AND EQUATES
    000353                             *                        *
    000354  01A2                       KSRBUF       RESV        4
    000355  01A6  2041                 LPBUF1       TEXT        ' A'
    000356  01A7                       LPBUF2       RESV        40
    000357  01CF                       CDRBUF       RESV        40
    000358  01F7  5443                 TERM         TEXT        'TC'
    000359  01F8                       COMBFR       RESV        36
    000360  021C  0707                 BELLS        TEXT        Z'0707'
    000361  021D  434F                 MSG1         TEXT        'COMMAND ERROR!'
            021E  4D4D
            021F  414E
            0220  4420
            0221  4552
            0222  524F
            0223  5221
    000362  0224  414C                 MSG2         TEXT        'ALL DONE'
            0225  4C20
            0226  444F
            0227  4E45
    000363  0228  4445                 MSG3         TEXT        'DEVICE ERROR'
            0229  5649
            022A  4345
            022B  2045
            022C  5252
            022D  4F52
    000364  022E                       TIMER        RESV        8
    000365  0236  434F                 QUES         TEXT        'COMMAND:'
            0237  4D4D
            0238  414E
            0239  443A
    000366  023A  494E                 QUES2        TEXT        'INPUT:'
            023B  5055
            023C  543A
    000367  023D  4341                 CH2          TEXT        'CA'
    000368  023E  5244                 CH4          TEXT        'RD'
    000369  023F  494E                 CH6          TEXT        'IN'
    000370  0240  2020                 BLANKS       DC          X'2020'
    000371  0241  2F20                 SLASH        TEXT        '/ '
    000372  0242  3A20                 COLON        TEXT        ': '
    000373                             *                        *
    000374                             *                        *
    000375                             *                        *
    000376  0243  0F01 FFF9            FINIS        NOP         CH2
    000377  0245  0000                              HLT
    000378  0246                                    END         COM200
0000 ERR COUNT
```

ACTION
CLM ACTION DURING LOADING, 3-13
ELACT COMMAND (END LOAD COMMAND), A-13
LACT COMMAND (LOAD ACTION), A-17

ACTIVATE
ACTIVATE LEVEL COMMAND (AL), 4-5

ADDRESS
ASSEMBLY LANGUAGE START ADDRESS DEFINITION, 3-17
COBOL LANGUAGE START ADDRESS DEFINITION, 3-18
ELOC COMMAND (DEFINE ADDRESS SYMBOL), A-14
FORTRAN LANGUAGE START ADDRESS DEFINITION, 3-17

ADMOD
ADMOD COMMAND (ADD LOAD MODULE), A-4

AL
ACTIVATE LEVEL COMMAND (AL), 4-5

AR
ALL REGISTERS COMMAND (AR), 4-5

AREA(S)
DATA STRUCTURE AREAS, 2-19
ESTABLISHING OVERLAY AREAS, 2-21
TSA COMMAND (TRAP SAVE AREA DEFINITION), A-23

ASSEMBLY
ASSEMBLY LANGUAGE START ADDRESS DEFINITION, 3-17

ASSIGN
ASSIGN COMMAND (AS), 4-5

ATFILE
ATFILE COMMAND (ATTACH FILE), A-4

ATLRN
ATLRN COMMAND (ATTACH LRN), A-5

ATTACH
ATFILE COMMAND (ATTACH FILE), A-4
ATLRN COMMAND (ATTACH LRN), A-5

BES
BES SOFTWARE FOR APPLICATION DEVELOPMENT (TBL), 2-3
BES SOFTWARE FOR APPLICATION EXECUTION (TBL), 2-2
OVERVIEW OF BES SOFTWARE SERVICES, 2-1

BINARY
BINARY SYNCHRONOUS COMMUNICATIONS (BSC 2780), 2-29

BOOTSTRAP
BOOTSTRAP RECORD FOR NONSTOP CLM LOADING (TBL), 3-8

BREAKPOINT(S)
LIST ALL BREAKPOINTS COMMAND (L*), 4-9
LIST BREAKPOINT COMMAND (LN), 4-10
SET BREAKPOINT COMMAND (SN), 4-11

BSC
BINARY SYNCHRONOUS COMMUNICATIONS (BSC 2780), 2-29
BSC 2780 COMMAND, A-6

BUFFER
FILE MANAGER BUFFER HANDLING, 2-10
SELECTING FILE AND BUFFER MANAGEMENT TECHNIQUES, 2-8

BUFFERED
BUFFERED READ OPERATIONS, 2-10
BUFFERED WRITE OPERATIONS, 2-11

BUFSPACE
BUFSPACE COMMAND (POOL DEFINITIONS), A-7

BUILDING
BUILDING, 3-1
BUILDING A CLM COMMAND FILE, 3-12
BUILDING AN ONLINE APPLICATION - PROCESS DIAGRAM (FIG), 3-2
PLANNING AND BUILDING WITH EXECUTIVE OBJECT MODULES, B-1

CALCULATIONS
SIZE CALCULATIONS FOR SYSTEM DATA STRUCTURES, 2-6

CH
CHANGE MEMORY COMMAND (CH), 4-6

CLEAR
CLEAR COMMAND (Cn), 4-6
CLEAR COMMAND (C*), 4-5

CLM
BOOTSTRAP RECORD FOR NONSTOP CLM LOADING (TBL), 3-8
BUILDING A CLM COMMAND FILE, 3-12
CLM ACTION DURING LOADING, 3-13
CLM FUNCTIONAL GROUPS COMPONENT MODULES AND RELATED COMMANDS (TBL), 3-7
CLM LOAD MODULE ORDER FOR PAPER TAPE (TBL), 3-11
EFFECTS OF CLM PARAMETERS ON MEMORY USAGE (TBL), 2-5
HOW TO INCLUDE OPTIONAL CLM EXTENSIONS, 3-6
INFORMATION FOR SYSTEM DATA STRUCTURES FROM CLM COMMANDS, 2-4

CLM (CONT)
INPUT DEVICES FOR CLM, A-3
PREPARING TO USE CLM, 3-1
SUMMARY OF CLM COMMANDS AND COMMAND FUNCTIONS (TBL), A-1

CLOCK
CLOCK COMMAND (SYSTEM CLOCK), A-8
REAL-TIME CLOCK (RTC), 4-17

Cn
CLEAR COMMAND (Cn), 4-6

COBOL
COBOL LANGUAGE START ADDRESS DEFINITION, 3-18

CODE
LINKING ORDER FOR CODE TEXT, 3-4

CODING
OVERLAY CODING CONVENTIONS, 2-21

COMM
COMM (COMMUNICATIONS SYSTEM COMMAND), A-9

COMMAND
ACTIVATE LEVEL COMMAND (AL), 4-5
ADMOD COMMAND (ADD LOAD MODULE), A-4
ALL REGISTERS COMMAND (AR), 4-5
ASSIGN COMMAND (AS), 4-5
ATFILE COMMAND (ATTACH FILE), A-4
ATLRN COMMAND (ATTACH LRN), A-5
BSC 2780 COMMAND, A-6
BUFSPACE COMMAND (POOL DEFINITIONS), A-7
BUILDING A CLM COMMAND FILE, 3-12
CHANGE MEMORY COMMAND (CH), 4-6
CLEAR COMMAND (Cn), 4-6
CLEAR COMMAND (C*), 4-5
CLOCK COMMAND (SYSTEM CLOCK), A-8
COMM (COMMUNICATIONS SYSTEM COMMAND), A-9
COMMAND FORMAT, A-2
DATE COMMAND (DATE AND TIME), A-9
DEBUGGING COMMAND FORMAT AND SYMBOLOGY, 4-3
DEBUGGING COMMAND LANGUAGE, 4-2
DEFINE COMMAND (Dn), 4-6
DEFINE TRACE COMMAND (DT), 4-7
DEVFILE COMMAND (FILE MANAGEMENT DEVICES), A-10
DEVICE COMMAND (I/O DEVICE TASK), A-11
DISPLAY MEMORY COMMAND (DH), 4-7
DUMP MEMORY COMMAND (DP), 4-7
ELACT COMMAND (END LOAD ACTION), A-13
ELOC COMMAND (DEFINE ADDRESS SYMBOL), A-14
EQLRN COMMAND (EQUATE LRN'S), A-14

COMMAND (CONT)
EVAL COMMAND (DEFINE VALUE SYMBOL), A-14
EXECUTE COMMAND (EN), 4-8
FILMGR COMMAND (FILE MANAGER), A-15
FMDISK COMMAND (FILE MANAGEMENT DISK), A-15
GO COMMAND (GO), 4-8
IOS COMMAND (I/O STREAM), A-15
LACT COMMAND (LOAD ACTION), A-17
LINE LENGTH COMMAND (LL), 4-9
LIST ALL BREAKPOINTS COMMAND (L*), 4-9
LIST BREAKPOINT COMMAND (Ln), 4-10
LOADING FROM DISK USING THE COMMAND PROCESSOR, 3-9
LTP DEFINITION COMMAND, A-17
LTPN COMMAND, A-18
MODEM DEFINITION COMMAND, A-19
OIM COMMAND (OPERATOR INTERFACE MANAGER DEFINITION), A-20
PRINT COMMAND (Pn), 4-10
PRINT COMMAND (P*), 4-10
PRINT HEADER LINE COMMAND (Hn), 4-8
PRINT HEXADECIMAL VALUE COMMAND (VH), 4-12
PRINT TRACE COMMAND (PT), 4-10
QUIT COMMAND (INITIATE LOADING), A-20
RESET FILE COMMAND (RF), 4-10
*COMMAND (COMMENTS), A-25
SET BREAKPOINT COMMAND (Sn), 4-11
SET LEVEL COMMAND (SL), 4-12
SET TEMPORARY LEVEL COMMAND (TL), 4-12
SPECIFY FILE COMMAND (SF), 4-11
STATION COMMAND, A-21
SUMMARY OF CLM COMMANDS AND COMMAND FUNCTIONS (TBL), A-1
SYMBOLS USED IN DEBUGGING COMMAND LINES (TBL), 4-4
SYS COMMAND (SYSTEM), A-21
TASK COMMAND (DEFINE TASK), A-22
TRAP COMMAND (TRAP VECTOR), A-22
TSA COMMAND (TRAP SAVE AREA DEFINITION), A-23
TTY COMMAND, A-23
VIP COMMAND, A-24

COMMANDS
CLM FUNCTIONAL GROUPS COMPONENT MODULES AND RELATED COMMANDS (TBL), 3-7
CONFIGURATION COMMANDS FOR SAMPLE COMMUNICATIONS APPLICATION, C-5
CONFIGURATION COMMANDS FOR SAMPLE INPUT/OUTPUT APPLICATION, C-1
CONFIGURATION LOAD MANAGER COMMANDS, A-1
DEBUGGING COMMANDS, 4-5
INFORMATION FOR SYSTEM DATA STRUCTURES FROM CLM COMMANDS, 2-4
LINK COMMANDS FOR SAMPLE COMMUNICATIONS PROGRAM, C-6

COMMANDS (CONT)
LINK COMMANDS FOR SAMPLE INPUT/ OUTPUT PROGRAM, C-1
SUMMARY OF CLM COMMANDS AND COMMAND FUNCTIONS (TBL), A-1
SUMMARY OF DEBUGGING COMMANDS BY FUNCTION (TBL), 4-2

COMMENTS
*COMMAND (COMMENTS), A-25

COMMUNICATIONS
BINARY SYNCHRONOUS COMMUNICATIONS (BSC 2780), 2-29
COMM (COMMUNICATIONS SYSTEM COMMAND), A-9
COMMUNICATIONS PLANNING, 2-27
CONFIGURATION COMMANDS FOR SAMPLE COMMUNICATIONS APPLICATION, C-5
LINK COMMANDS FOR SAMPLE COMMUNICATIONS PROGRAM, C-6
PRIORITY LEVEL REQUIREMENTS FOR COMMUNICATIONS, 2-27
REQUESTING COMMUNICATIONS FUNCTIONS, 2-28
SAMPLE COMMUNICATIONS PROGRAM, C-6

COMPONENT
CLM FUNCTIONAL GROUPS COMPONENT MODULES AND RELATED COMMANDS (TBL), 3-7

CONFIGURATION
APPLICATION CONFIGURATION AND LOADING, 3-8
APPLICATION CONFIGURATION EXAMPLE, C-1
CONFIGURATION COMMANDS FOR SAMPLE COMMUNICATIONS APPLICATION, C-5
CONFIGURATION COMMANDS FOR SAMPLE INPUT/OUTPUT APPLICATION, C-1
CONFIGURATION LOAD MANAGER COMMANDS, A-1
MEMORY LAYOUT DURING CONFIGURATION (FIG), 3-14
USING THE CONFIGURATION LOAD MANAGER (STAGE 6), 3-6

CONSOLE
LOADING FROM DISK WITH AN OPERATOR'S CONSOLE, 3-9
LOADING FROM DISK WITHOUT AN OPERATOR'S CONSOLE, 3-10

CONVENTIONS
OVERLAY CODING CONVENTIONS, 2-21
PRINTER SPACE CONVENTIONS, 2-12

CREATING
CREATING EXECUTIVE LOAD MODULES, B-1

CREATION
LOAD MODULE CREATION (STAGE 5), 3-4
OBJECT MODULE CREATION (STAGE 4), 3-4
SOURCE MODULE CREATION AND EDITING (STAGES 2 AND 3), 3-3

C*
CLEAR COMMAND (C*), 4-5

CURRENT
CURRENT LOAD MODULE MEMORY LAYOUT (FIG), 3-5

DATA
DATA STRUCTURE AREAS, 2-19
DATA STRUCTURES, 4-18
HARDWARE/EXECUTIVE DATA STRUCTURES (FIG), 4-18
INFORMATION FOR SYSTEM DATA STRUCTURES FROM CLM COMMANDS, 2-4
MEMORY DATA STRUCTURES (FIG), 2-20
SIZE CALCULATIONS FOR SYSTEM DATA STRUCTURES, 2-6

DATE
DATE COMMAND (DATE AND TIME), A-9

DEBUG
ADDITIONAL OPERATING NOTES FOR THE ONLINE DEBUG PROGRAM, 4-14
ONLINE DEBUG PROGRAM FUNCTIONS, 4-2
USING THE ONLINE DEBUG PROGRAM, 4-1

DEBUGGING
DEBUGGING, 4-1
DEBUGGING COMMAND FORMAT AND SYMBOLOGY, 4-3
DEBUGGING COMMAND LANGUAGE, 4-2
DEBUGGING COMMANDS, 4-5
DEBUGGING DURING ONLINE APPLICATION DEVELOPMENT, 4-16
SUMMARY OF DEBUGGING COMMANDS BY FUNCTION (TBL), 4-2
SYMBOLS USED IN DEBUGGING COMMAND LINES (TBL), 4-4
USING THE ONLINE DEBUGGING PROGRAM, 4-13

DEDICATED
HARDWARE DEDICATED LOCATIONS, 2-19

DEFINE
DEFINE COMMAND (DN), 4-6
DEFINE TRACE COMMAND (DT), 4-7
ELOC COMMAND (DEFINE ADDRESS SYMBOL), A-14
EVAL COMMAND (DEFINE VALUE SYMBOL), A-14
TASK COMMAND (DEFINE TASK), A-22

DEFINING
DEFINING APPLICATION DESIGN OBJECTIVES, 2-3
DEFINING ONLINE ENVIRONMENT CHARACTERISTICS, 2-4

DEFINITION(S)
ASSEMBLY LANGUAGE START ADDRESS DEFINITION, 3-17
BUFSPACE COMMAND (POOL DEFINITIONS), A-7
COBOL LANGUAGE START ADDRESS DEFINITION, 3-18
FORTRAN LANGUAGE START ADDRESS DEFINITION, 3-17
LTP DEFINITION COMMAND, A-17
MODEM DEFINITION COMMAND, A-19
OIM COMMAND (OPERATOR INTERFACE MANAGER DEFINITION), A-20
TSA COMMAND (TRAP SAVE AREA DEFINITION), A-23

DESIGN
DEFINING APPLICATION DESIGN OBJECTIVES, 2-3

DESIGNING
DESIGNING PROGRAMS FOR AN ONLINE ENVIRONMENT, 2-13

DEVFILE
DEVFILE COMMAND (FILE MANAGEMENT DEVICES), A-10

DEVICE
DEVICE COMMAND (I/O DEVICE TASK), A-11

DEVICES
DEVFILE COMMAND (FILE MANAGEMENT DEVICES), A-10
INPUT DEVICES FOR CLM, A-3

DH
DISPLAY MEMORY COMMAND (DH), 4-7

DIAGRAM
BUILDING AN ONLINE APPLICATION - PROCESS DIAGRAM (FIG), 3-2

DISK
FMDISK COMMAND (FILE MANAGEMENT DISK), A-15
LOAD AND HALT PROCEDURES FOR DISK, 3-9
LOADING FROM DISK USING THE COMMAND PROCESSOR, 3-9
LOADING FROM DISK WITH AN OPERATOR'S CONSOLE, 3-9
LOADING FROM DISK WITHOUT AN OPERATOR'S CONSOLE, 3-10

DISPLAY
DISPLAY MEMORY COMMAND (DH), 4-7

Dn
DEFINE COMMAND (Dn), 4-6

DP
DUMP MEMORY COMMAND (DP), 4-7

DRIVERS
INPUT AND OUTPUT DRIVERS, 2-18

DT
DEFINE TRACE COMMAND (DT), 4-7

DUMP
DUMP MEMORY COMMAND (DP), 4-7

EDITING
SOURCE MODULE CREATION AND EDITING (STAGES 2 AND 3), 3-3

ELACT
ELACT COMMAND (END LOAD ACTION), A-13

ELOC
ELOC COMMAND (DEFINE ADDRESS SYMBOL), A-14

EMULATION
IBM 2780 REMOTE TERMINAL EMULATION, 2-29

En
EXECUTE COMMAND (En), 4-8

END
ELACT COMMAND (END LOAD ACTION), A-13

ENVIRONMENT
DEFINING ONLINE ENVIRONMENT CHARACTERISTICS, 2-4
DESIGNING PROGRAMS FOR AN ONLINE ENVIRONMENT, 2-13

EQLRN
EQLRN COMMAND (EQUATE LRN'S), A-14

EQUATE
EQLRN COMMAND (EQUATE LRN'S), A-14

ERRORS
HANDLING LOAD ERRORS, 4-20

EVAL
EVAL COMMAND (DEFINE VALUE SYMBOL), A-14

EXECUTE
EXECUTE COMMAND (En), 4-8

EXECUTION
BES SOFTWARE FOR APPLICATION EXECUTION (TBL), 2-2
MEMORY LAYOUT DURING APPLICATION EXECUTION (FIG), 3-16
SERVICES AVAILABLE FOR APPLICATION EXECUTION, 2-1

EXECUTIVE
CREATING EXECUTIVE LOAD MODULES, B-1
EXECUTIVE OBJECT MODULES (TBL), B-1
PLANNING AND BUILDING WITH EXECUTIVE OBJECT MODULES, B-1
SELECTING EXECUTIVE MODULES, 2-8

EXTENSIONS
HOW TO INCLUDE OPTIONAL CLM EXTENSIONS, 3-6

FILE
ATFILE COMMAND (ATTACH FILE), A-4
BUILDING A CLM COMMAND FILE, 3-12
DEVFILE COMMAND (FILE MANAGEMENT DEVICES), A-10
FILE MANAGER BUFFER HANDLING, 2-10
FILMGR COMMAND (FILE MANAGER), A-15
FMDISK COMMAND (FILE MANAGEMENT DISK), A-15
HOW TO ESTIMATE OVERLAY FILE SIZE, 2-25
INTERACTIVE FILE TYPE/LFN COORDINATION, 2-12
LEVEL 6-TO-LEVEL 6 FILE TRANSMISSION, 2-29
MEMORY AND WORK FILE SPACE USAGE (TBL), 4-1
OUTPUT FILE PREALLOCATION (STAGE 1), 3-3
RESET FILE COMMAND (RF), 4-10
SELECTING FILE AND BUFFER MANAGEMENT TECHNIQUES, 2-8
SPECIFY FILE COMMAND (SF), 4-11

FILMGR
FILMGR COMMAND (FILE MANAGER), A-15

FLOATABLE
EXAMPLE OF FLOATABLE OVERLAYS, 2-24

FMDISK
FMDISK COMMAND (FILE MANAGEMENT DISK), A-15

FORMAT
COMMAND FORMAT, A-2
DEBUGGING COMMAND FORMAT AND SYMBOLOGY, 4-3

FORTRAN
FORTRAN LANGUAGE START ADDRESS DEFINITION, 3-17

FUNCTION(S)
SUMMARY OF DEBUGGING COMMANDS BY FUNCTION (TBL), 4-2

FUNCTIONAL
CLM FUNCTIONAL GROUPS, COMPONENT MODULES AND RELATED COMMANDS (TBL), 3-7
ONLINE DEBUG PROGRAM FUNCTIONS, 4-2

FUNCTIONAL (CONT)
REQUESTING COMMUNICATIONS FUNCTIONS, 2-28
SUMMARY OF CLM COMMANDS AND COMMAND FUNCTIONS (TBL), A-1

GO
GO COMMAND (GO), 4-8

HALT
LOAD AND HALT PROCEDURES FOR DISK, 3-9

HARDWARE
HARDWARE DEDICATED LOCATIONS, 2-19

HARDWARE/EXECUTIVE
HARDWARE/EXECUTIVE DATA STRUCTURES (FIG), 4-18

HEADER
PRINT HEADER LINE COMMAND (HN), 4-8

HEXADECIMAL
PRINT HEXADECIMAL VALUE COMMAND (VH), 4-12

Hn
PRINT HEADER LINE COMMAND (Hn), 4-8

HONEYWELL-SUPPLIED
NAMES AND SIZES OF HONEYWELL-SUPPLIED LOAD MODULES (TBL), 2-9

IBM
IBM 2780 REMOTE TERMINAL EMULATION, 2-29

INITIALIZATION
INITIALIZATION PROCESSING (FIG), B-2
INITIALIZATION SUBROUTINES, 2-25
NEW INITIALIZATION MODULES (FIG), B-3
REGISTER USE BY SYSTEM INITIALIZATION SUBROUTINES (TBL), 2-26

INITIATE
QUIT COMMAND (INITIATE LOADING), A-20

INPUT
INPUT AND OUTPUT DRIVERS, 2-18
INPUT DEVICES FOR CLM, A-3

INPUT/OUTPUT
CONFIGURATION COMMANDS FOR SAMPLE INPUT/OUTPUT APPLICATION, C-1
LINK COMMANDS FOR SAMPLE INPUT/OUTPUT PROGRAM, C-1
SAMPLE INPUT/OUTPUT PROGRAM, C-1
SELECTING INPUT/OUTPUT MODULES, 2-8

INTERACTIVE
INTERACTIVE FILE TYPE/LFN COORDINATION, 2-12

INTERFACE
OIM COMMAND (OPERATOR INTERFACE MANAGER DEFINITION), A-20

IOS
IOS COMMAND (I/O STREAM), A-15

I/O
DEVICE COMMAND (I/O DEVICE TASK), A-11
IOS COMMAND (I/O STREAM), A-15

LACT
LACT COMMAND (LOAD ACTION), A-17

LANGUAGE
ASSEMBLY LANGUAGE START ADDRESS DEFINITION, 3-17
COBOL LANGUAGE START ADDRESS DEFINITION, 3-18
DEBUGGING COMMAND LANGUAGE, 4-2
FORTRAN LANGUAGE START ADDRESS DEFINITION, 3-17

LENGTH
LINE LENGTH COMMAND (LL), 4-9

LEVEL(S)
ACTIVATE LEVEL COMMAND (AL), 4-5
ATTACHING LRN'S TO LEVELS, 2-16
LEVEL 6-TO-LEVEL 6 FILE TRANSMISSION, 2-29
PRIORITY LEVELS, 2-13
PRIORITY LEVEL REQUIREMENTS FOR COMMUNICATIONS, 2-27
RELATIVE PRIORITY LEVEL ASSIGNMENTS (TBL), 2-14
SAMPLE LRN PRIORITY LEVEL ATTACHMENTS (FIG), 2-15
SAMPLE STATEMENTS ATTACHING LRN'S TO LEVELS (FIG), 2-16
SET LEVEL COMMAND (SL), 4-12
SET TEMPORARY LEVEL COMMAND (TL), 4-12

LINE(S)
LINE LENGTH COMMAND (LL), 4-9
PRINT HEADER LINE COMMAND (HN), 4-8
SYMBOLS USED IN DEBUGGING COMMAND LINES (TBL), 4-4

LINK
LINK COMMANDS FOR SAMPLE COMMUNICATIONS PROGRAM, C-6
LINK COMMANDS FOR SAMPLE INPUT/ OUTPUT PROGRAM, C-1

LINKING
LINKING ORDER FOR CODE TEXT, 3-4

LIST
LIST ALL BREAKPOINTS COMMAND (L*), 4-9
LIST BREAKPOINT COMMAND (Ln), 4-10

LL
LINE LENGTH COMMAND (LL), 4-9

Ln
LIST BREAKPOINT COMMAND (Ln), 4-10

LOAD
ADMOD COMMAND (ADD LOAD MODULE), A-4
CLM LOAD MODULE ORDER FOR PAPER TAPE (TBL), 3-11
CONFIGURATION LOAD MANAGER COMMANDS, A-1
CREATING EXECUTIVE LOAD MODULES, B-1
CURRENT LOAD MODULE MEMORY LAYOUT (FIG), 3-5
FLACT COMMAND (END LOAD ACTION), A-13
HANDLING LOAD ERRORS, 4-20
LACT COMMAND (LOAD ACTION), A-17
LOAD AND HALT PROCEDURES FOR DISK, 3-9
LOAD MODULE CREATION (STAGE 5), 3-4
LOCATING LOAD MODULES, 4-15
NAMES AND SIZES OF HONEYWELL-SUPPLIED LOAD MODULES (TBL), 2-9
SUMMARY OF LOAD MODULE PREPARATION, 3-6
USING THE CONFIGURATION LOAD MANAGER (STAGE 6), 3-6

LOADING
APPLICATION CONFIGURATION AND LOADING, 3-8
BOOTSTRAP RECORD FOR NONSTOP CLM LOADING (TBL), 3-8
CLM ACTION DURING LOADING, 3-13
LOADING FROM DISK USING THE COMMAND PROCESSOR, 3-9
LOADING FROM DISK WITH AN OPERATOR'S CONSOLE, 3-9
LOADING FROM DISK WITHOUT AN OPERATOR'S CONSOLE, 3-10
LOADING FROM PAPER TAPE, 3-10
MEMORY LAYOUT AFTER LOADING (FIG), 3-15
NONSTOP APPLICATION LOADING, 3-8
QUIT COMMAND (INITIATE LOADING), A-20

LOCATING
LOCATING LOAD MODULES, 4-15

LOCATIONS
HARDWARE DEDICATED LOCATIONS, 2-19

LOGICAL
LOGICAL RESOURCE NUMBERS, 2-15
PHYSICAL AND LOGICAL RESOURCE REQUIREMENTS (TBL), 2-3

LRN'S
ATTACHING LRN'S TO LEVELS, 2-16
EQLRN COMMAND (EQUATE LRN'S), A-14
SAMPLE STATEMENTS ATTACHING LRN'S TO LEVELS (FIG), 2-16

LRN
ATLRN COMMAND (ATTACH LRN), A-5
SAMPLE LRN PRIORITY LEVEL ATTACHMENTS (FIG), 2-15

L*
LIST ALL BREAKPOINTS COMMAND (L*), 4-9

LTP
LTP DEFINITION COMMAND, A-17

LTPn
LTPn COMMAND, A-18

MANAGEMENT
DEVFILE COMMAND (FILE MANAGEMENT DEVICES), A-10
FMDISK COMMAND (FILE MANAGEMENT DISK), A-15
SELECTING FILE AND BUFFER MANAGEMENT TECHNIQUES, 2-8

MANAGER
CONFIGURATION LOAD MANAGER COMMANDS, A-1
FILE MANAGER BUFFER HANDLING, 2-10
FILMGR COMMAND (FILE MANAGER), A-15
OIM COMMAND (OPERATOR INTERFACE MANAGER DEFINITION), A-20
USING THE CONFIGURATION LOAD MANAGER (STAGE 6), 3-6

MEMORY
CHANGE MEMORY COMMAND (CH), 4-6
CURRENT LOAD MODULE MEMORY LAYOUT (FIG), 3-5
DISPLAY MEMORY COMMAND (DH), 4-7
DUMP MEMORY COMMAND (DP), 4-7
EFFECTS OF CLM PARAMETERS ON MEMORY USAGE (TBL), 2-5
MEMORY AND WORK FILE SPACE USAGE (TBL), 4-1
MEMORY DATA STRUCTURES (FIG), 2-20
MEMORY LAYOUT AFTER LOADING (FIG), 3-15
MEMORY LAYOUT DURING APPLICATION EXECUTION (FIG), 3-16
MEMORY LAYOUT DURING CONFIGURATION (FIG), 3-14
MEMORY USAGE CONSIDERATIONS, 2-19

MODEM
MODEM DEFINITION COMMAND, A-19

MODULE
ADMOD COMMAND (ADD LOAD MODULE), A-4
CLM IOAD MODULE ORDER FOR PAPER TAPE (TBL), 3-11
CURRENT LOAD MODULE MEMORY LAYOUT (FIG), 3-5
LOAD MODULE CREATION (STAGE 5), 3-4
OBJECT MODULE CREATION (STAGE 4), 3-4

MODULE (CONT)
SOURCE MODULE CREATION AND EDITING (STAGES 2 AND 3), 3-3
SUMMARY OF LOAD MODULE PREPARATION, 3-6

MODULES
CLM FUNCTIONAL GROUPS COMPONENT MODULES AND RELATED COMMANDS (TBL), 3-7
CREATING EXECUTIVE LOAD MODULES, B-1
EXECUTIVE OBJECT MODULES (TBL), B-1
LOCATING LOAD MODULES, 4-15
NAMES AND SIZES OF HONEYWELL-SUPPLIED LOAD MODULES (TBL), 2-9
NEW INITIALIZATION MODULES (FIG), B-3
PLANNING AND BUILDING WITH EXECUTIVE OBJECT MODULES, B-1
SELECTING EXECUTIVE MODULES, 2-8
SELECTING INPUT/OUTPUT MODULES, 2-8

MONITOR
MONITOR POINTS, 4-16

MULTITASKING
MULTITASKING, 2-13

NONFLOATABLE
EXAMPLE OF NONFLOATABLE OVERLAYS, 2-22

NONSTOP
BOOTSTRAP RECORD FOR NONSTOP CLM LOADING (TBL), 3-8
NONSTOP APPLICATION LOADING, 3-8

OBJECT
EXECUTIVE OBJECT MODULES (TBL), B-1
OBJECT MODULE CREATION (STAGE 4), 3-4
PLANNING AND BUILDING WITH EXECUTIVE OBJECT MODULES, B-1

OBJECTIVES
DEFINING APPLICATION DESIGN OBJECTIVES, 2-3

OIM
OIM COMMAND (OPERATOR INTERFACE MANAGER DEFINITION), A-20

ONLINE
ADDITIONAL OPERATING NOTES FOR THE ONLINE DEBUG PROGRAM, 4-14
BUILDING AN ONLINE APPLICATION - PROCESS DIAGRAM (FIG), 3-2
DEBUGGING DURING ONLINE APPLICATION DEVELOPMENT, 4-16
DEFINING ONLINE ENVIRONMENT CHARACTERISTICS, 2-4
DESIGNING PROGRAMS FOR AN ONLINE ENVIRONMENT, 2-13
ONLINE DEBUG PROGRAM FUNCTIONS, 4-2
STARTING AN ONLINE APPLICATION, 3-17

ONLINE (CONT)
USING THE ONLINE DEBUG PROGRAM, 4-1
USING THE ONLINE DEBUGGING PROGRAM, 4-13

OPERATING
ADDITIONAL OPERATING NOTES FOR THE ONLINE DEBUG PROGRAM, 4-14

OPERATIONS
BUFFERED READ OPERATIONS, 2-10
BUFFERED WRITE OPERATIONS, 2-11

OPERATOR'S
LOADING FROM DISK WITH AN OPERATOR'S CONSOLE, 3-9
LOADING FROM DISK WITHOUT AN OPERATOR'S CONSOLE, 3-10

OPERATOR
OIM COMMAND (OPERATOR INTERFACE MANAGER DEFINITION), A-20

OPTIONAL
HOW TO INCLUDE OPTIONAL CLM EXTENSIONS, 3-6

ORDER
CLM LOAD MODULE ORDER FOR PAPER TAPE (TBL), 3-11
LINKING ORDER FOR CODE TEXT, 3-4

OUTPUT
INPUT AND OUTPUT DRIVERS, 2-18
OUTPUT FILE PREALLOCATION (STAGE 1), 3-3
SAMPLE ZXMAP OUTPUT (FIG), 4-16

OVERLAYS
ESTABLISHING OVERLAY AREAS, 2-21
EXAMPLE OF FLOATABLE OVERLAYS, 2-24
EXAMPLE OF NONFLOATABLE OVERLAYS, 2-22
HOW TO ESTIMATE OVERLAY FILE SIZE, 2-25
OVERLAY CODING CONVENTIONS, 2-21
OVERLAY PLANNING, 2-19

OVERVIEW
OVERVIEW OF BES SOFTWARE SERVICES, 2-1

PAPER
CLM LOAD MODULE ORDER FOR PAPER TAPE (TBL), 3-11
LOADING FROM PAPER TAPE, 3-10

PARAMETERS
EFFECTS OF CLM PARAMETERS ON MEMORY USAGE (TBL), 2-5

PHYSICAL
PHYSICAL AND LOGICAL RESOURCE REQUIREMENTS (TBL), 2-3

PLANNING
COMMUNICATIONS PLANNING, 2-27
OVERLAY PLANNING, 2-19
PLANNING, 2-1
PLANNING AND BUILDING WITH EXECUTIVE OBJECT MODULES, B-1

Pn
PRINT COMMAND (Pn), 4-10

POOL
BUFSPACE COMMAND (POOL DEFINITIONS), A-7

PREALLOCATION
OUTPUT FILE PREALLOCATION (STAGE 1), 3-3

PREPARATION
SUMMARY OF LOAD MODULE PREPARATION, 3-6

PREPARING
PREPARING TO USE CLM, 3-1

PRINT
PRINT COMMAND (Pn), 4-10
PRINT COMMAND (P*), 4-10
PRINT HEADER LINE COMMAND (Hn), 4-8
PRINT HEXADECIMAL VALUE COMMAND (VH), 4-12
PRINT TRACE COMMAND (PT), 4-10

PRINTER
PRINTER SPACE CONVENTIONS, 2-12

PRIORITY
PRIORITY LEVEL REQUIREMENTS FOR COMMUNICATIONS, 2-27
PRIORITY LEVELS, 2-13
RELATIVE PRIORITY LEVEL ASSIGNMENTS (TBL), 2-14
SAMPLE LRN PRIORITY LEVEL ATTACHMENTS (FIG), 2-15

PROCESSING
INITIALIZATION PROCESSING (FIG), B-2

PROCESSOR
LOADING FROM DISK USING THE COMMAND PROCESSOR, 3-9

PROGRAM
ADDITIONAL OPERATING NOTES FOR THE ONLINE DEBUG PROGRAM, 4-14
LINK COMMANDS FOR SAMPLE COMMUNICATIONS PROGRAM, C-6
LINE COMMANDS FOR SAMPLE INPUT/OUTPUT PROGRAM, C-1
ONLINE DEBUG PROGRAM FUNCTIONS, 4-2
SAMPLE COMMUNICATIONS PROGRAM, C-6
SAMPLE INPUT/OUTPUT PROGRAM, C-1
USING THE ONLINE DEBUG PROGRAM, 4-1
USING THE ONLINE DEBUGGING PROGRAM, 4-13

PROGRAMS
DESIGNING PROGRAMS FOR AN ONLINE ENVIRONMENT, 2-13

P*
PRINT COMMAND (P*), 4-10

PT
PRINT TRACE COMMAND (PT), 4-10

QUIT
QUIT COMMAND (INITIATE LOADING), A-20

READ
BUFFERED READ OPERATIONS, 2-10

REAL-TIME
REAL-TIME CLOCK (RTC), 4-17

RECORD
BOOTSTRAP RECORD FOR NONSTOP CLM LOADING (TBL), 3-8

REGISTER(S)
ALL REGISTERS COMMAND (AR), 4-5
REGISTER USE BY SYSTEM INITIALIZATION SUBROUTINES (TBL), 2-26

RELATIVE
RELATIVE PRIORITY LEVEL ASSIGNMENTS (TBL), 2-14

REMOTE
IBM 2780 REMOTE TERMINAL EMULATION, 2-29

REQUESTING
REQUESTING COMMUNICATIONS FUNCTIONS, 2-28
REQUESTING TASKS, 2-17

RESET
RESET FILE COMMAND (RF), 4-10

RESOURCE
LOGICAL RESOURCE NUMBERS, 2-15
PHYSICAL AND LOGICAL RESOURCE REQUIREMENTS (TBL), 2-3

RF
RESET FILE COMMAND (RF), 4-10

RTC
REAL-TIME CLOCK (RTC), 4-17

SAMPLE
CONFIGURATION COMMANDS FOR SAMPLE COMMUNICATIONS APPLICATION, C-5
CONFIGURATION COMMANDS FOR SAMPLE INPUT/OUTPUT APPLICATION, C-1

SAMPLE (CONT)
LINK COMMANDS FOR SAMPLE COMMUNICATIONS PROGRAM, C-6
LINK COMMANDS FOR SAMPLE INPUT/OUTPUT PROGRAM, C-1
SAMPLE COMMUNICATIONS PROGRAM, C-6
SAMPLE INPUT/OUTPUT PROGRAM, C-1
SAMPLE LRN PRIORITY LEVEL ATTACHMENTS (FIG), 2-15
SAMPLE STATEMENTS ATTACHING LRN'S TO LEVELS (FIG), 2-16
SAMPLE ZXMAP OUTPUT (FIG), 4-16

SAVE
TSA COMMAND (TRAP SAVE AREA DEFINITION), A-23

SELECTING
SELECTING EXECUTIVE MODULES, 2-8
SELECTING FILE AND BUFFER MANAGEMENT TECHNIQUES, 2-8
SELECTING INPUT/OUTPUT MODULES, 2-8
SELECTING SYSTEM VARIABLES, 2-4

SERVICES
OVERVIEW OF BES SOFTWARE SERVICES, 2-1
SERVICES AVAILABLE FOR APPLICATION DEVELOPMENT, 2-1
SERVICES AVAILABLE FOR APPLICATION EXECUTION, 2-1

SET
SET BREAKPOINT COMMAND (Sn), 4-11
SET LEVEL COMMAND (SL), 4-12
SET TEMPORARY LEVEL COMMAND (TL), 4-12

SF
SPECIFY FILE COMMAND (SF), 4-11

SIZE
HOW TO ESTIMATE OVERLAY FILE SIZE, 2-25
SIZE CALCULATIONS FOR SYSTEM DATA STRUCTURES, 2-6

SIZES
NAMES AND SIZES OF HONEYWELL-SUPPLIED LOAD MODULES (TBL), 2-9

SL
SET LEVEL COMMAND (SL), 4-12

SN
SET BREAKPOINT COMMAND (Sn), 4-11

SOFTWARE
BES SOFTWARE FOR APPLICATION DEVELOPMENT (TBL), 2-3
BES SOFTWARE FOR APPLICATION EXECUTION (TBL), 2-2
OVERVIEW OF BES SOFTWARE SERVICES, 2-1

SOURCE
SOURCE MODULE CREATION AND EDITING (STAGES 2 AND 3), 3-3

SPACE
MEMORY AND WORK FILE SPACE USAGE (TBL), 4-1
PRINTER SPACE CONVENTIONS, 2-12

SPECIFY
SPECIFY FILE COMMAND (SF), 4-11

START
ASSEMBLY LANGUAGE START ADDRESS DEFINITION, 3-17
COBOL LANGUAGE START ADDRESS DEFINITION, 3-18
FORTRAN LANGUAGE START ADDRESS DEFINITION, 3-17

STARTING
STARTING AN ONLINE APPLICATION, 3-17

STATION
STATION COMMAND, A-21

STREAM
IOS COMMAND (I/O STREAM), A-15

STRUCTURE(S)
DATA STRUCTURE AREAS, 2-19
DATA STRUCTURES, 4-18
HARDWARE/EXECUTIVE DATA STRUCTURES (FIG), 4-18
INFORMATION FOR SYSTEM DATA STRUCTURES FROM CLM COMMANDS, 2-4
MEMORY DATA STRUCTURES (FIG), 2-20
SIZE CALCULATIONS FOR SYSTEM DATA STRUCTURES, 2-6

SUBROUTINES
INITIALIZATION SUBROUTINES, 2-25
REGISTER USE BY SYSTEM INITIALIZATION SUBROUTINES (TBL), 2-26

SUMMARY
SUMMARY OF CLM COMMANDS AND COMMAND FUNCTIONS (TBL), A-1
SUMMARY OF DEBUGGING COMMANDS BY FUNCTION (TBL), 4-2
SUMMARY OF LOAD MODULE PREPARATION, 3-6

SYMBOL(S)
ELOC COMMAND (DEFINE ADDRESS SYMBOL), A-14
EVAL COMMAND (DEFINE VALUE SYMBOL), A-14
EXTERNALLY DEFINED SYMBOLS, 3-4
SYMBOLS USED IN DEBUGGING COMMAND LINES (TBL), 4-4

SYMBOLOGY
DEBUGGING COMMAND FORMAT AND SYMBOLOGY, 4-3

SYNCHRONOUS
BINARY SYNCHRONOUS COMMUNICATIONS (BSC 2780), 2-29

SYS
SYS COMMAND (SYSTEM), A-21

SYSTEM
CLOCK COMMAND (SYSTEM CLOCK), A-8
COMM (COMMUNICATIONS SYSTEM COMMAND), A-9
INFORMATION FOR SYSTEM DATA STRUCTURES FROM CLM COMMANDS, 2-4
REGISTER USE BY SYSTEM INITIALIZATION SUBROUTINES (TBL), 2-26
SELECTING SYSTEM VARIABLES, 2-4
SIZE CALCULATIONS FOR SYSTEM DATA STRUCTURES, 2-6
SYS COMMAND (SYSTEM), A-21

TAPE
CLM LOAD MODULE ORDER FOR PAPER TAPE (TBL), 3-11
LOADING FROM PAPER TAPE, 3-10

TASK(S)
DEVICE COMMAND (I/O DEVICE TASK), A-11
REQUESTING TASKS, 2-17
TASK COMMAND (DEFINE TASK), A-22

TECHNIQUES
SELECTING FILE AND BUFFER MANAGEMENT TECHNIQUES, 2-8

TERMINAL
IBM 2780 REMOTE TERMINAL EMULATION, 2-29

TEXT
LINKING ORDER FOR CODE TEXT, 3-4

TIME
DATE COMMAND (DATE AND TIME), A-9

TL
SET TEMPORARY LEVEL COMMAND (TL), 4-12

TRACE
DEFINE TRACE COMMAND (DT), 4-7
PRINT TRACE COMMAND (PT), 4-10
TRACE HISTORY, 4-19

TRANSMISSION
LEVEL 6-TO-LEVEL 6 FILE TRANSMISSION, 2-29

TRAP
TRAP COMMAND (TRAP VECTOR), A-22
TSA COMMAND (TRAP SAVE AREA DEFINITION), A-23

TSA
TSA COMMAND (TRAP SAVE AREA DEFINITION), A-23

TTY
TTY COMMAND, A-23

VALUE
FVAL COMMAND (DEFINE VALUE SYMBOL), A-14
PRINT HEXADECIMAL VALUE COMMAND (VH), 4-12

VARIABLES
SELECTING SYSTEM VARIABLES, 2-4

VECTOR
TRAP COMMAND (TRAP VECTOR), A-22

VH
PRINT HEXADECIMAL VALUE COMMAND (VH), 4-12

VIP
VIP COMMAND, A-24

WRITE
BUFFERED WRITE OPERATION, 2-11

ZXMAP
SAMPLE ZXMAP OUTPUT (FIG), 4-16

HONEYWELL INFORMATION SYSTEMS

Technical Publications Remarks Form

TITLE	SERIES 60 (LEVEL 6) GCOS/BES2 PLANNING AND BUILDING AN ONLINE APPLICATION	ORDER NO.	AU49, REV. 0
		DATED	JULY 1976

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 ______________________

____

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

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.

16297, 1876, Printed in U.S.A.

AU49, Rev. 0