Honeywell

LEVEL 6

SOFTWARE

GCOS 6 MOD 400 SYSTEM CONCEPTS

SERIES 60 (LEVEL 6)

GCOS 6 MOD 400

SYSTEM CONCEPTS

SUBJECT

System Concepts for Series 60 (Level 6) GCOS 6 MOD 400 Operating System Software

SOFTWARE SUPPORTED

This publication supports Release 0100 of Series 60 (Level 6) GCOS 6 MOD 400 Operating System; see the Manual Directory in this manual for a list of other manuals that support the release.

ORDER NUMBER

CB20, Rev. 0

January 1978

Honeywell

PREFACE

This manual presents a general description of Honeywell-supplied software and system concepts for the Series 60 (Level 6) GCOS 6 MOD 400 operating system. Unless otherwise stated, the term GCOS refers to the GCOS 6 MOD 400 software. The term Level 6 refers to the Series 60 (Level 6) hardware on which the software executes.

Section 1 summarizes system features, provides information on how to use manuals in the document set, and lists the contents of other manuals in the set.

Section 2 describes the GCOS software facilities.

Section 3 contains details on the characteristics and use of the file system.

Section 4 summarizes system configuration procedures, methods of accessing the system, and the functions and features of the command environment.

Section 5 discusses the execution environment, presenting descriptions of task groups and tasks, memory pools, and bound units.

Section 6 describes features related to task execution, including priority level assignments, task coordination, and system handling of executing tasks.

Section 7 summarizes the functions of the Data Entry Facility, the Remote Batch Facility, and the File Transmission Facility.

Appendix A describes the facilities available to provide compatibility for using GCOS/BES and GCOS/BES2 programs and files with GCOS 6 MOD 400 systems.

Appendix B presents GCOS programming conventions.

Appendix C describes the Level 6 hardware resources and includes an equipment requirements list.

Appendix D is a glossary.

File No.: 1S13

CB20

MANUAL DIRECTORY

The following publications comprise the GCOS 6 manual set and support Release 0100 of the GCOS 6 MOD 400 system software.

Order No.	*Manual Title*
CB01	*GCOS 6 Program Preparation*
CB01A	*Addendum A*
CB02	*GCOS 6 Commands*
CB03	*GCOS 6 Communications Processing*
CB04	*GCOS 6 Sort/Merge*
CB05	*GCOS 6 Data File Organizations and Formats*
CB06	*GCOS 6 System Messages*
CB07	*GCOS 6 Assembly Language Reference*
CB08	*GCOS 6 System Service Macro Calls*
CB09	*GCOS 6 RPG Reference*
CB10	*GCOS 6 Intermediate COBOL Reference*
CB20	*GCOS 6 MOD 400 System Concepts*
CB21	*GCOS 6 MOD 400 Program Execution and Checkout*
CB22	*GCOS 6 MOD 400 Programmer's Guide*
CB23	*GCOS 6 MOD 400 System Building*
CB24	*GCOS 6 MOD 400 Operator's Guide*
CB25	*GCOS 6 MOD 400 FORTRAN Reference*
CB26	*GCOS 6 MOD 400 Entry-Level COBOL Reference*
CB30	*Remote Batch Facility User's Guide*
CB31	*Data Entry Facility User's Guide*
CB33	*Level 6/Level 6 File Transmission*
CB34	*Level 6/Level 62 File Transmission*
CB35	*Level 6/Level 64 (Release 0300) File Transmission*
CB36	*Level 6/Level 66 File Transmission*
CB37	*Level 6/Series 200/2000 File Transmission*
CB38	*Level 6/BSC 2780 File Transmission*
CB39	*Level 6/Level 64 (Release 0220) File Transmission*

In addition, the following documents provide general hardware information:

Order No.	*Manual Title*
AS22	*Honeywell Level 6 Minicomputer Handbook*
AT04	*Level 6 System and Peripherals Operation Manual*

CONTENTS

ILLUSTRATIONS

TABLES

SECTION 1
SYSTEM CHARACTERISTICS

GCOS 6 MOD 400 software is a disk-based operating system that supports multitasking, real-time, or data communications applications in one or more online streams. In addition, program development or other batch type applications can be performed concurrently in a single batch stream.

GCOS is a multifunctional system, capable of providing a variety of processing functions: development and execution of applications, forms data entry, file transmission to other computers, and serving as a remote batch terminal for a host processor. The system can be configured to process different functional applications concurrently. For example, a user can run his own applications, utilize other system functionality such as the data collection capability, and communicate with a host processor, all at the same time.

SOFTWARE FEATURES

The operating system includes Monitor, File System, and data communications facilities. The Monitor supports the execution of user application tasks and provides a set of system services that enables users to control execution of individual tasks and to synchronize multiple tasks with one another and with time-related events. The Monitor controls the loading of user programs and manages requests for available memory. It provides standard system trap handling routines for responding to exception conditions and also allows users to provide their own trap handling routines for user-caused trap conditions.

The File System software offers an extensive set of logical I/O access methods. It provides device-independent access to any device for sequential files, and direct access to disk files. In addition the File System software automatically manages the space utilization of mounted disk volumes, thus allowing users to create, expand, and release disk files as required by online applications needs.

The operating system offers two levels of communications interface. Remote/local terminals may be accessed through the sequential file interface of file management, or for more direct control of the communications environment, by using the systems physical I/O interface. The communications facility includes line protocol handlers for teleprinter and VIP devices and binary synchronous communications (BSC).

The software includes a powerful and comprehensive set of program preparation components, utilities and debugging aids for applications developments, all running under control of the Monitor. Programming languages include assembly language, RPG, FORTRAN, and Entry-Level and Intermediate COBOL. Commercial Instruction Processor (CIP) and Scientific Instruction Processor (SIP) hardware or simulators are available with the system. The RPG and Intermediate COBOL Compilers generate code for the CIP: the FORTRAN Compiler generates SIP code. The Assembler supports both CIP and SIP instructions.

The system supports concurrent execution of a variety of functions that interface with the GCOS communications software, including the File Transmission Facility, Remote Batch Facility, and Data Entry Facility. These components permit the GCOS system to be connected to Honeywell and other host processor, all in a real-time, distributed systems environment.

The file transmission capability (invoked by means of utility programs) supports transmission of files between the Level 6 and Series 60 (Level 6, 62, 64, or 66) or Series 2000 processors, using the polled VIP protocol; or between the Level 6 and non-Honeywell processors, using the BSC protocol.

The Remote Batch Facility (RBF) permits a Level 6 system to be used for job submission and output delivery for one or more Series 60 (Level 66) or Series 6000 host processors, using the Remote Computer Interface (RCI) or High-Level Data Link (HDLC) protocols.

Local processing (such as program development and user application execution) can occur concurrently with remote batch processing.

The Data Entry Facility (DEF) provides a data collection capability that includes creation/modification of forms; formatted data input; validation, extraction and verification of data; and formatted printing of data. The facility supports multiple independent operator display stations that use VIP devices.

GCOS 6 MOD 400 facilities accommodate applications developed under the GCOS/Basic Executive Systems (BES). These facilities include utilities to move source and object program files between the two operating systems, and support of BES system service calls via a special interface package. Files created under the BES File Manager are supported directly by MOD 400; programs created under BES must be relinked under MOD 400.

OPERATING FEATURES

The operating system supports concurrent execution of multiple tasks (sequences of instructions that perform identifiable functions) running under one or more task groups. Each task group owns the resources necessary for execution of an application program (one or more related tasks). The task group runs independently in its own operating environment while sharing the resources of the operating system.

Multiprogramming can be achieved by defining more than one application task group to be run concurrently. Serial execution of tasks in a task group can be accomplished by stepping through execution of the tasks in sequence; multitasking can be achieved by causing tasks in the group to be executed concurrently.

Multiple online task groups can be run concurrently with a single batch task group. The batch task group (used for program development or a batch-oriented user application) can be rolled out to a disk to obtain additional memory for online applications.

The number of task groups being run is limited only by the amount of memory available. Concurrently executing task groups may occupy independent, dedicated memory areas, or they may contend for space within a memory pool. When one task group is deleted, the released memory is available to other task groups in the same pool. The Monitor allocates memory dynamically from pools and can relocate programs at load time. Once a task group requests execution, it is dispatched according to its assigned priority level. When multiple tasks share a priority level, they are serviced in a round-robin fashion.

Use of the file system by multiple independent users is facilitated by the arrangement of file system entries (directories and files) in a tree-structured hierarchy. Each directory or file is identified by a pathname that indicates the path from the root directory of the hierarchical structure to the particular directory or file. File reference is simplified through the use of pathnames relative to a working directory that indicates a user's current position in the file system hierarchy. Access to sharable files and devices is controlled by file attributes and concurrency procedures.

SUMMARY OF SYSTEM FEATURES

The GCOS 6 software offers the following capabilities:

- Provides a multi-user operating system
- Supports multiple concurrent programming environments, with applications being run in one batch stream and one or more online streams
- Controls program preparation through the operating system
- Handles program preparation and execution of user applications concurrently
- Handles execution of multiple user applications
- Permits multitask execution within each user application
- Controls the execution sequence of user applications
- Supports real-time operations
- Provides communications support
- Is time and interrupt driven
- Allows device independent programming
- Supports program overlay capability

- o Provides four programming languages: assembly language, FORTRAN, RPG, and two levels of COBOL (entry level and intermediate)
- o Provides a hierarchical file system with extensive utility support
- o Supports four disk file organizations
- o Supports code sharing via reentrant programs
- o Permits multiple functions that interface with communications facilities to be run concurrently with application development and execution
- o Supports file transmission between the Level 6 and other computers
- o Provides the Remote Batch Facility, permitting the Level 6 system to be used for job submission to a host processor
- o Provides the Data Entry Facility, permitting forms creation and data collection
- o Provides compatibility software for GCOS/BES and GCOS/BES2 programs and files

GUIDE TO USING THE MANUAL SET

This guide to the manuals is arranged according to functions that might be performed by an applications programmer, a systems programmer, or an operator. As used in this guide, the applications programmer writes applications programs; the system programmer configures the system and defines the environment for each application; and the operator operates the system from the operator terminal. These functions could be performed by three different persons or by the same person serving in the different capacities.

Applications Programmer's Manual Guide

Figure 1-1 illustrates the suggested sequence for using the munuals. If you wish to start using the system by writing an application program, begin by using the *Programmer's Guide* manual. It illustrates: (1) various ways to gain access to the system, (2) a sample Editor session, and (3) for application languages, the procedure for performing program preparation and execution. Working with the small subset of system commands used within examples is a good approach to learning the system command set. This approach for getting started assumes that a system programmer has already configured and started up a suitable application environment. While using the system, you may wish to familiarize yourself with the system facilities described in the *System Concepts* manual.

Through examples, the *Programmer's Guide* illustrates how to use the system facilities. Other manuals provide reference material. The *Program Preparation* manual contains Editor directives (statements) to create and update an application language source unit. For each of the languages, the appropriate language reference manual contains the description of the language statements. Operating system dependencies, if any, that affect how you write the application are described in the *Programmer's Guide*. If the application uses communications, refer to the *Communications Processing* manual. Read the *Data File Organizations and Formats* manual if you require a better understanding of a language-supported file organization that is to be used in an application or if you must calculate the size of a data file. You can use Monitor macro calls, as described in the *System Service Macro Calls* manual, in assembly language programs. Before your program can be entered for execution, it must be linked as described in the *Program Execution and Checkout* manual.

For program compilation or assembly and execution, the procedures described in the *Programmer's Guide* might be sufficient. To obtain more control over the execution of your program or utilize the system facilities more completely or efficiently, use the commands described in the *Commands* manual. If you wish to use the operator terminal, read the *Operator's Guide* to learn how to use that terminal. In many cases, the description of commands must be supplemented by system concepts described in the *System Concepts* manual. Rather than read all the conceptual material at one time, you may find it more meaningful to refer to it in conjunction with the appropriate reference material. The *Commands* manual also describes the utilities. The Patch, Debug, and Dump utilities are described in the *Program Execution and Checkout* manual; file transmission from Level 6 to a host system is described in the appropriate *File Transmission* manual. Error messages and return status codes are listed in the *System Messages* manual.

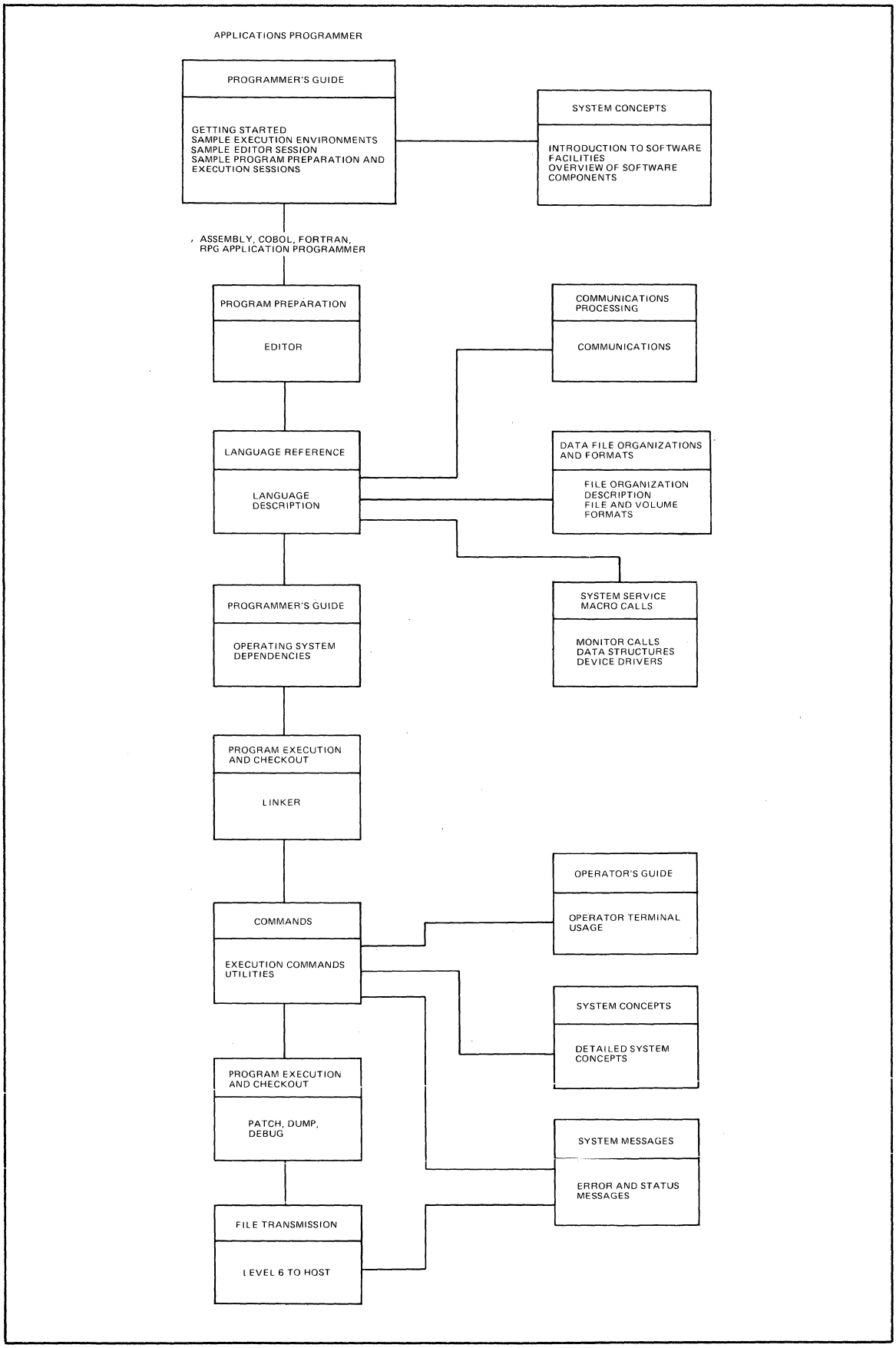


Figure 1-1. Applications Programmer's Guide to Manuals

System Programmer's Manual Guide

Figure 1-2 illustrates the suggested sequence for using the manuals. The *System Building* manual provides you with the configuration directives (statements) and startup procedures to configure and start up a MOD 400, a Remote Batch Facility (RBF), or a Data Entry Facility (DEF) system. You must know the conceptual material in the *System Concepts* manual in order to successfully use the configuration directives. To tailor an applications environment suitable for the intended application, you use operator commands described in the *Operator's Guide* manual. Error messages are listed in the *System Messages* manual. If you are working with an application that runs under the BES operating system, the *System Concepts* manual contains MOD 400 and BES compatibility considerations.

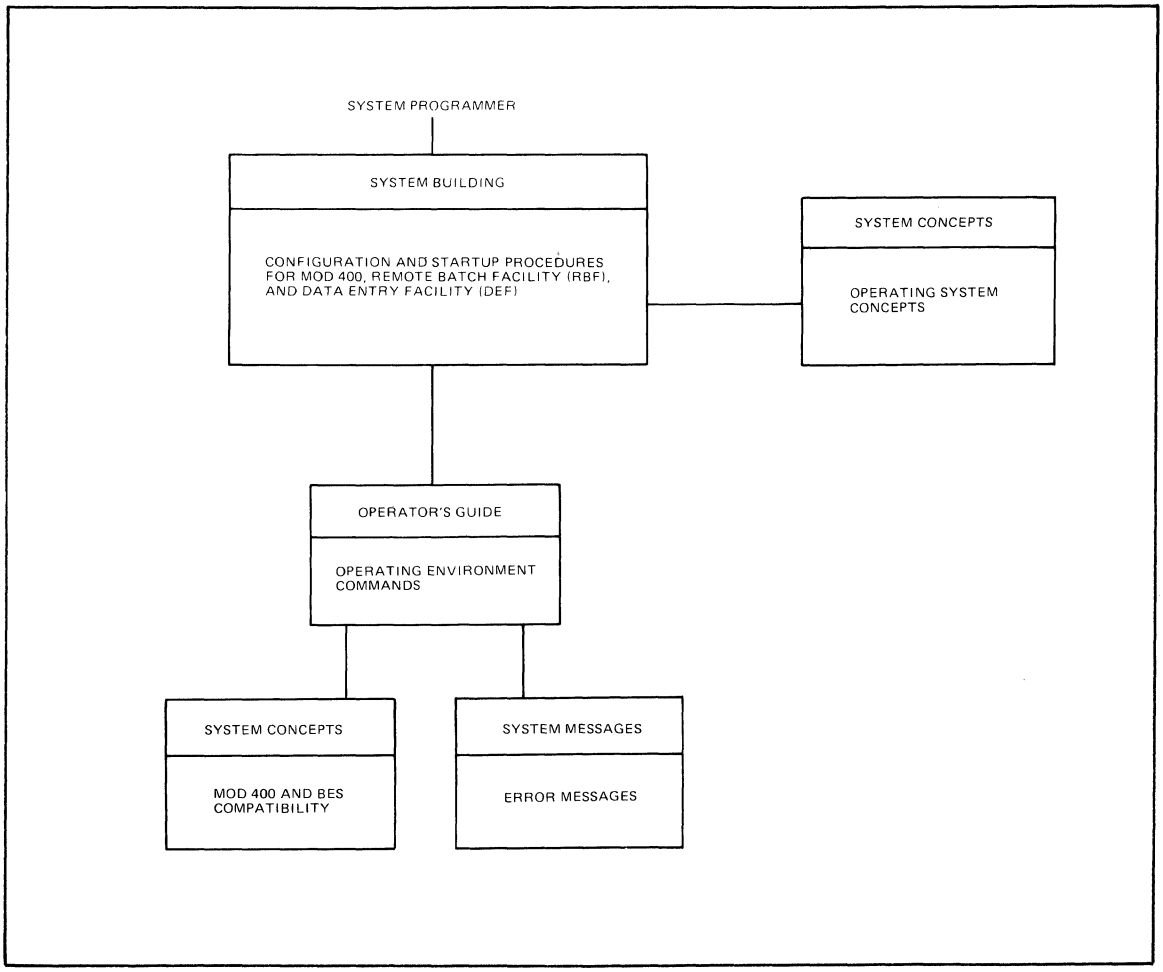


Figure 1-2. System Programmer's Guide to Manuals

Operator's Manual Guide

Figure 1-3 illustrates the suggested sequence for using the manuals. Specific operator job functions must be determined by each installation; a large system might have a person assigned as an operator; a small system might have each programmer also act as an operator. The *Operator's Guide* indicates those system procedures performed through the operator terminal and describes operator commands used in system operation.

The *Programmer's Guide* contains examples using commands (described in the *Commands* manual) that are similar to operator commands. The *System Concepts* manual provides an understanding of the operating system. Note that the *Operator's Guide* describes using the operator terminal for operator functions to enter operator commands to the system task group, or for user functions to enter commands to a user task group. To run the utilities, use the commands (described in the *Commands* manual) entered through the operator terminal functioning as a user terminal. Error messages are listed in the *System Message* manual.

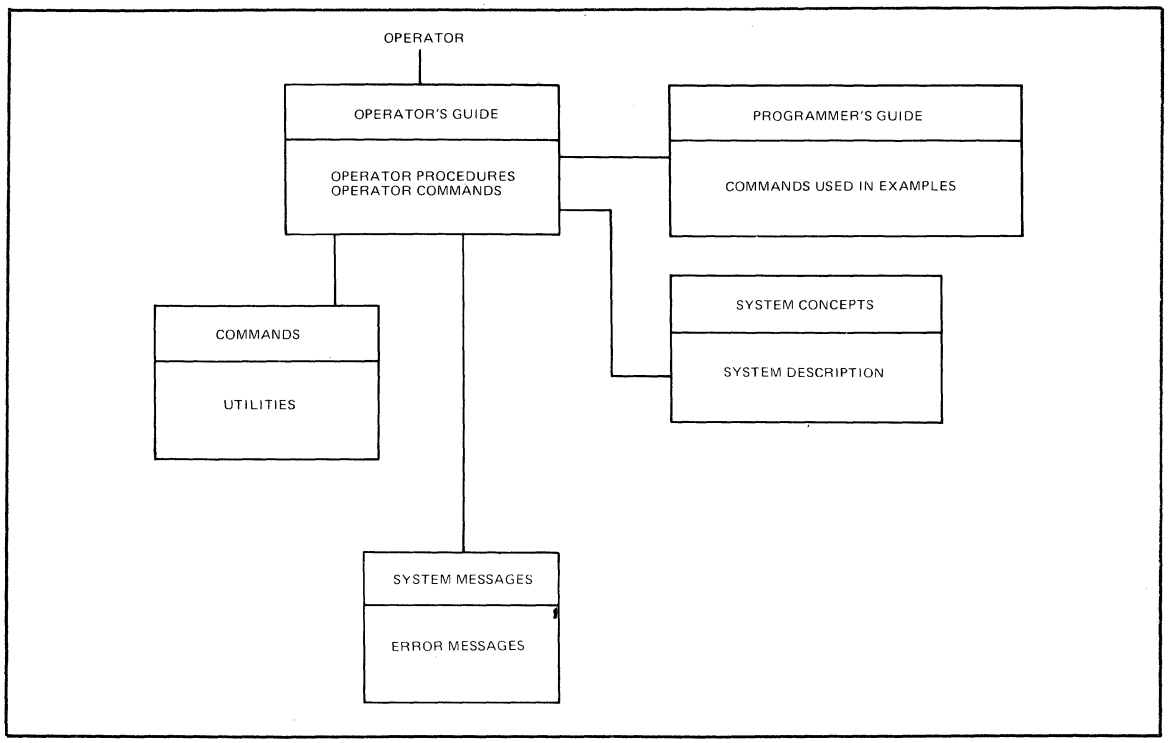


Figure 1-3. Operator's Guide to Manuals

RBF- and DEF-User's Manual Guide

Figure 1-4 illustrates the suggested sequence for using the manuals. The system programmer configuration functions have been done and the system is ready to be used for Remote Batch Facility (RBF) functions or Data Entry Facility (DEF) functions. The *Programmer's Guide* manual provides sample login execution environments similar to ones that might be at your facility. The *Remote Batch Facility User's Guide* is used for RBF operations and the *Data Entry Facility User's Guide* is used for DEF operations.

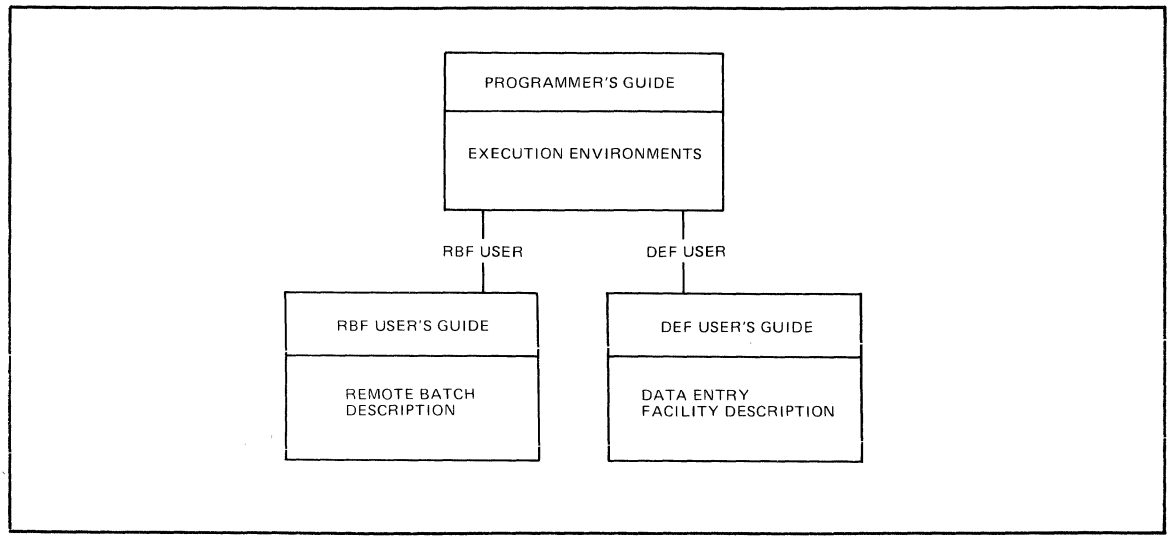


Figure 1-4. RBF- and DEF-User's Guide To Manual

SOFTWARE DOCUMENT SET

This *System Concepts* manual briefly describes GCOS software, system features, and operating concepts. Most of the background information needed to use the reference material in other manuals of this set is presented in Section 3 through 6 of this manual. Except

for summaries, this material is not duplicated in other manuals and covers the following subjects:

- o Task groups and tasking
- o Memory definition and use
- o Operating system features
- o File system and communications concepts
- o Operating environment configuration

This manual is the sole source for reference material on the compatibility of GCOS/BES1 and BES2 programs and files with a MOD 400 system. Programming conventions are presented in Appendix B of this manual, and a glossary of GCOS 6 MOD 400 terms is in Appendix D.

The contents of other documents in the manual set are summarized briefly below.

- o *GCOS 6 Program Preparation,* Order No. CB01 – Overview of the programming steps to prepare a program for execution. Suffix conventions for files used in program preparation. Detailed description of Editor. Rules for writing assembly language programs using SLIC (SAF/LAF independent code).
- o *GCOS 6 Commands,* Order No. CB02 – Description of command line format, task interrupt break function, activating an application program, and extending the command set. Detailed description of commands, utilities, and language processor execution. Description of additional command line arguments, terminal characteristics at login, Intersystem Link (ISL) directives, and File Change directives; ASCII and EBCDIC character sets.
- o *GCOS 6 Communications Processing,* Order No. CB03 – Introduction to communications software. Description of communications processing through COBOL, assembly language, File System and FORTRAN; sample communications programs; Dump MLCP (DUMCP) utility; TTY, VIP, and BSC control characters; ASCII and EBCDIC character sets.
- o *GCOS 6 Sort/Merge*, Order No. CB04 – Description of the Sort and Merge program features, statement formats, and report contents. Includes file and memory requirements, operating procedures, sample programs, using Sort as a subroutine, debug mode execution, and ASCII collating sequence.
- o *GCOS 6 Data File Organizations and Formats*, Order No. CB05 – Description of disk and magnetic tape data file organizations support for application programs; disk and magnetic tape record, file, and volume formats; unit record file formats; file and volume headers; ASCII and EBCDIC character sets.
- o *GCOS 6 System Messages*, Order No. CB06 – Description of messages reported by system components. Procedure for adding user messages.
- o *GCOS 6 Assembly Language Reference*, Order No. CB07 – Complete description of all instructions, instruction formats, control statements, types of data handled, and macro language statements. Description of Scientific Instruction Processor (SIP) and Commercial Instruction Processor (CIP) instructions.
- o *GCOS 6 System Service Macro Calls*, Order No. CB08 – Description of macro call syntax, register and addressing conventions. Detailed description of system services macro calls for the Monitor and File System and for defining data structures; physical I/O device drivers; Trap Handler; Monitor and File System data structures; writing a user device driver; contents of registers for system service macro calls; ASCII and EBCDIC character set.
- o *GCOS 6 RPG Reference*, Order No. CB09 – Complete description of RPG data processing including: a primer on RPG programming, RPG specification form entries, description and use of the RPG fixed logic cycle, and operating instructions with sample programs.
- o *GCOS 6 Intermediate COBOL Reference*, Order No. CB10 – Complete description of the general features of Intermediate COBOL programs, language elements, language syntax, the four major divisions of an Intermediate COBOL program, specific format

descriptions of all Intermediate COBOL statements (including programming examples incorporating each statement), and the types of files and data handled, compiler diagnostics, ASCII collating sequence, COBOL glossary, comparison of standard COBOL with Intermediate COBOL, and Intermediate COBOL run-time considerations.

- *GCOS 6 MOD 400 Program Execution and Checkout*, Order No. CB21 – Overview of program execution sequence. Detailed descriptions of Linker, Debug, Patch, Dump Memory (MDUMP), Dump Edit (DPEDIT), and interpreting and using memory dumps. Table of system service macro calls ordered by function code.
- *GCOS 6 MOD 400 Programmer's Guide*, Order No. CB22 – Description of various possible programming environments at an installation and the ways to access the system for each environment. Sample Editor session. Examples illustrating how to prepare and execute COBOL, FORTRAN, SORT and assembly language programs; how to call FORTRAN routines from an Entry-Level COBOL main program; and FORTRAN chaining. Explanation of headers on listings.
- *GCOS 6 MOD 400 System Building*, Order No. CB23 – Description of system configuration and startup procedures for the MOD 400 operating system, the Data Entry Facility (DEF), and the Remote Batch Facility (RBF); configuration directives; system disk layout; system overlays; minimum system hardware and configuration requirements to do program preparation; and startup halts. Description of procedures to transfer files to system disk; create a single diskette system; place a shared version of a utility in the system library; and load and execute the Intersystem Link (ISL) loader for ISL configuration.
- *GCOS 6 MOD 400 Operator's Guide*, Order No. CB24 – Description of routine system startup, activation of the login capability, system operator interface with the system (OIM), operator commands; task interrupt break function from the operator terminal; additional operator command line arguments; listener component setup for login capability; system halt conditions; ASCII character set.
- *GCOS 6 MOD 400 FORTRAN Reference*, Order No. CB25 – Complete description of all statements, instruction formats, types of files and data handled, FORTRAN run-time support routines (intrinsic functions, tasking, I/O), and compiler diagnostics.
- *GCOS 6 MOD 400 Entry-Level COBOL Reference*, Order No. CB26 – Complete description of the general features of Entry-Level COBOL programs, language elements, language syntax, the four major divisions of an Entry-Level COBOL program, specific format descriptions of all Entry-Level COBOL statements (including programming examples incorporating each statement), and the types of files and data handled, compiler diagnostics, ASCII collating sequence, COBOL glossary, comparison of standard COBOL with Entry-Level COBOL, and Entry-Level COBOL run-time considerations.
- *Remote Batch Facility User's Guide*, Order No. CB30 – Description of remote batch operations: communicating with the host, preparing job decks, managing job streams, input and output processing, operator commands and messages, host software control records.
- *Data Entry Facility User's Guide*, Order No. CB31 – Operation of the Data Entry Facility. Description of operation of the Operator Display Station; forms and table development; data entry and verification process; file printing; system supervisory and utility operations; interfacing with data entry and applications programs; and error and system messages.
- *Level 6/Level 6 File Transmission*, Order No. CB33
- *Level 6/Level 62 File Transmission*, Order No. CB34
- *Level 6/Level 64 (Release 0300) File Transmission*, Order No. CB35
- *Level 6/Level 66 File Transmission*, Order No. CB36
- *Level 6/Series 200/2000 File Transmission*, Order No. CB37
- *Level 6/BSC2780 File Transmission*, Order No. CB38
- *Level 6/Level 64 (Release 0220) File Transmission*, Order No. CB39

Each of the above documents describes the capability of the particular file transmission facility, including file organizations supported, code sets, line protocols, and

equipment requirements. Individual sections of the manuals provide the operating information necessary to perform a file transfer from either end of a network (Level 6 and host).

- o *Honeywell Level 6 Minicomputer Handbook*, Order No. AS22 – Descriptions of hardware models, central processor, processor architecture and features, instruction set, registers, peripheral devices, control panel, software, various controllers and system features, as well as maintenance and site preparation information.
- o *Level 6 System and Peripherals Operation*, Order No. AT04 – Description of the operation of the models 6/30 and 6/40 control panels, the operation of each peripheral attachable to Level 6 hardware, and operator trouble-shooting procedures.

SECTION 2
SOFTWARE FACILITIES

GENERAL FEATURES OF SOFTWARE

The system provides a comprehensive array of software to perform multitasking; real-time and data communications applications; and batch, remote batch and data entry processing. The operating system controls execution of tasks and accessing of external devices and files. A complete set of program preparation software is available to develop and debug programs written in COBOL, FORTRAN, RPG or assembly language. An extensive set of utility programs is provided to support program development and execution, and transmission of files from the Level 6 to other computers. System software components are summarized in Figure 2-1.

OPERATING SYSTEM

- MONITOR
- FILE SYSTEM
- PHYSICAL I/O
- COMMUNICATIONS

PROGRAM PREPARATION

- EDITOR
- MACRO PREPROCESSOR
- ASSEMBLER
- FORTRAN COMPILER[a]
- ENTRY–LEVEL COBOL COMPILER[a]
- INTERMEDIATE COBOL COMPILER
- RPG COMPILER
- LINKER
- DEBUG

MOD 400–BES COMPATIBILITY[a]

- EMULATOR
- BUFFER MANAGER

UTILITY PROGRAMS

- COMPARE
- COPY
- CREATE FILE
- CREATE VOLUME
- DUMP EDIT
- DUMP MEMORY
- DUMP MLCP
- EXPORT PAM FILE
- FILE CHANGE
- FILE DUMP
- IMPORT PAM FILE
- ISL CONFIGURATOR
- LIST CREATION DATE
- LIST NAMES
- PATCH
- PRINT FILE
- RENAME FILE
- RESET MAP
- SORT/MERGE
- TRANSMIT FILE

MLCP SOFTWARE

- CHANNEL CONTROL PROGRAMS
- OFFLINE LOADER

SYSTEM CONTROL INTERFACES

- COMMANDS
- OPERATOR COMMANDS
- SYSTEM SERVICE MACRO CALLS

CONFIGURATION

- CONFIGURATION LOAD MANAGER
- HONEYWELL–SUPPLIED SYSTEM

RUN–TIME ROUTINES

- FORTRAN ROUTINES[a]
- ENTRY–LEVEL COBOL ROUTINES[a]
- INTERMEDIATE COBOL ROUTINES

HARDWARE SIMULATORS

- SINGLE PRECISION SCIENTIFIC (SIP) SIMULATOR
- DOUBLE AND SINGLE PRECISION (DSIP) SCIENTIFIC SIMULATOR
- COMMERCIAL INSTRUCTION PROCESSOR (CIP) SIMULATOR

REMOTE BATCH FACILITY

DATA ENTRY FACILITY

[a]These software components are available only with the SAF version of MOD 400.

Figure 2-1. GCOS Software

The software is available in a SAF (short address form) version, which supports up to 64K words of memory on model 6/34, 6/36 and 6/43 processors, and a LAF (long address form) version which supports up to 256K words of memory on model 6/43 processors. Hardware resources associated with this system are described in Appendix B.

INTERFACES TO OPERATING SYSTEM

The software supports the following control interfaces to the operating system:

- *Commands* submitted by a user to the command processor of the user task group
- *Operator commands* submitted by the operator to the command processor of the system task group
- *System service macro calls,* specified in assembly language programs, that invoke Monitor and file system services for user task groups

Command Language

There are five functional categories of commands:

- To control execution
- To control directories and files
- To invoke program preparation software
- To invoke utility software
- Interactive commands

Some control functions at the task group level are available through commands. Commands are described in the *Commands* manual.

Commands for Execution Control

Once an online task group is created, commands written by the user can be executed under the task group. More comprehensive control of execution is provided to the assembly language program through system service macro calls. Commands are used to:

- Create then initiate other online task groups, or spawn online task groups. This provides a multiprogramming capability.
- Abort or delete an online task group, or terminate the task group issuing the request. The abort and delete functions are not available through the batch task group.
- Create then initiate the execution of a sequence of tasks under an online task group, or spawn online tasks within an online group. Using this capability an application can be executed as a sequence of steps. When the sequencing is done so as to have several tasks active simultaneously, there is multitask execution in one task group.
- Initiate and control the execution of tasks in the batch task group except the creation of the batch task group.
- Control of external switches for intertask communication.
- List the status of all tasks or open files in a task group.
- Get or remove a file from reserved status.

Commands for Directory and File Control

The file system is based on a tree-structured directory hierarchy. To locate a file, the directory pathname must be known. In order to write programs that are independent of the pathname of the physical file, a program uses a logical file number (LFN). More comprehensive control of directories and files is provided to the assembly language program through system service macro calls.

Commands are used to:

- Create or release a directory or file
- List the pathname of the working directory; change the pathname of the working directory

- o List, in the order searched, the directories that are searched for a given pathname; list file entries in a specified disk directory
- o Modify the share, read, or write attributes of a disk file
- o Associate or dissociate a pathname with a logical file number

Commands for Program Preparation

Software to perform program preparation is invoked using a command. Component-specific arguments are provided in these commands. The command name is often identical to the software name, e.g., COBOL, FORTRAN, LINKER, RPG.

Commands for Utility Software Execution

A utility is invoked through a command. The command is often identical to the software name, e.g., PATCH, SORT, MERGE.

Interactive Commands

Interactive commands permit the user to:

- o Establish and terminate access to the system
- o Request execution of a batch task group
- o Send messages to the operator
- o Display the current time

Operator Commands

There are three functional categories of operator commands: execution control; directory, file and device control; and system operation monitoring. Operator commands operate on a task group level and cannot be used to control the execution sequence of tasks in the batch task group or in a user online task group. Operator commands are entered through the operator terminal or read from a command file. A description of the operator commands is found in the *Operator's Guide* manual.

Operator Commands for Execution Control

Initially, operator commands are used to define the operating environment. Subsequently, they can be used to control system operation from the operator terminal. Operator commands are employed to:

- o Create, initiate, abort or delete either a batch or online task group
- o Spawn an online task group
- o Temporarily suspend or reactivate an online task group
- o Temporarily suspend and roll out, or reactivate and roll in the batch task group
- o Load or unload a shareable bound unit from a system memory pool
- o Load assembled firmware files into writable control store (WCS)

Operator Commands **for** *Directory, File and Device Control*

Operator commands are used to:

- o Change a system library or working directory pathname
- o Modify the share, read, or write attributes of a disk file

Operator Commands to Monitor the System

Operator commands are used to:

- o List all task groups and requests queued for batch execution
- o List the status of all tasks or open files in a task group
- o List the pathname of the working directory
- o List, in the order searched, the directories that are searched for a given bound unit

System Service Macro Calls

System service macro calls are available to the assembly language program to perform a wide variety of Monitor and file system service functions, similar in some instances to those functions accessible through commands. There are two functional categories of system service macro calls: to control execution, and to control directories and files. The macro calls are described in the *System Service Macro Calls* manual.

Macro Calls for Execution Control

Monitor service macro calls are used to:

- Control task groups and tasks
- Manage memory allocation
- Load and execute overlays
- Coordinate the use of resources within an online task group through semaphores
- Control execution based on real-time clock considerations
- Enable or disable user traps
- Display or suppress the display of messages on the operator's terminal
- Designate a task group's command-in, user-in, error-out, and user-out files
- Communicate directly with device drivers to control input/output and devices
- Control external switches for intertask group communication
- Associate or dissociate a pathname with a logical file number.

Macro Calls for Directory and File Control

Monitor service macro calls are used to:

- Create or release a directory or file
- Change or obtain the pathname of the working directory
- Rename a file or directory
- Open, close, get (reserve), or remove the reservation of a file
- Lock/unlock records in a file
- Get information describing a file
- Test the status of an outstanding file activity
- Wait on list until input or output is complete
- Read, write, rewrite, or delete a record of a file
- Read from, or write a block to a file

OPERATING SYSTEM SOFTWARE

The operating system contains software for execution control, the file system, physical I/O, and communications.

Monitor Software

The Monitor contains software to execute requests for Monitor functions and to maintain the control tables that are necessary for the orderly processing of requests. These functions are obtained through commands, system service macro calls, and statements in higher-level languages. Monitor software includes:

- *Task manager* – Handles the disposition of tasks within the system, and responds to requests placed against tasks. It processes requests to activate tasks, returns control to interrupted tasks, and synchronizes, suspends and terminates tasks.
- *Clock manager* – Handles all requests to control tasks based on real-time considerations, and responds to requests for the time of day and date in ASCII format.
- *Memory Manager* – Controls dynamic requests for memory or the return of memory to a memory pool.
- *Trap manager* – Handles the transfer of execution control from an executing program to a predefined trap location when a trap (a special condition such as a hardware error) occurs. The trap manager handles system traps and allows a task group to connect its own trap routines for specific traps.

- o *Operator interface manager* – Manages all messages sent simultaneously by multiple task groups to the operator terminal or from the operator terminal to a task group.
- o *Loader* – Loads the root and overlays of a bound unit dynamically from a disk.
- o *Listener/login* – The listener monitors a selected set of local and remote terminals, reporting any change of state (for example, connect, disconnect) to the login component. If a user submits a login command requesting access to the system, the login component requests that a task be spawned for the user.
- o *Command processor* – Processes all commands. It is the lead task of the batch task group and can be the lead task of an online task group.

File System Software

The file system is based on a tree-structured hierarchy and software functions are provided to create or maintain this directory structure, locate a file by its pathname, create and maintain data files, control concurrent use of files, and provide for the logical transfer of records between an application program and an external device. These functions are available through commands or, for an assembly language program, through system service macro calls, described in the *Commands* manual and *System Service Macro Calls* manual, respectively.

The File System software handles input/output functions of each of the different supported devices, including communications. For disk, it provides four file organizations and access to them; namely, sequential, relative, indexed, and fixed relative file organizations. (Fixed relative file organization is compatible with BES and BES2 files.) Sequential access is provided for magnetic tape, communications, printer, card reader, and terminals. A description of the data file organizations and their properties is found in the *Data File Organizations and Formats* manual.

The languages COBOL, FORTRAN, and RPG use the logical file organizations listed above. The language reference manual for each language provides statements for accessing the logical files.

An assembly language program can access files through file and data management macro calls to the Monitor or through the physical I/O drivers; both methods are described in the *System Service Macro Calls* manual.

The interface to communications software is described in the *Communications Processing* manual.

Physical Input/Output Software

An assembly language program can use physical input/output driver software which works at the hardware physical level. Each peripheral and communications device type has a driver which is a reentrant procedure that can control one or more devices. A description of the peripheral drivers and the physical I/O macro calls is found in the *System Service Macro Calls* manual. Macro calls for communications are described in the *Communications Processing* manual.

Communications Software

Communications software is accessible through the standard input/output interface, is memory and MLCP resident, and interacts with Monitor software to process user communications applications. With the Honeywell-supplied communications software, users need not provide their own communications system programs.

The communications software is user-driven. It answers the phone in response to a user-issued connect; it polls terminals in response to user-issued reads. Users (application or system software) must provide buffers to the communications software to accommodate read and write operations.

Communications software provides a common I/O interface to its users through the standard physical I/O interface (the $RQIO macro call). The communications software components and their functions are summarized below.

- o *Communications supervisor* – Queues user service requests and activates the appropriate line protocol handler, interacts with the user program through system software when

a transaction is complete, and services connect/disconnect requests and line protocol handler timeouts.

- *Phone monitor* – Provides data set control for detection of phone connect/disconnect, and provides the capability of "hanging up" the phone connection upon user request.
- *Line protocol handler* (LPH) – Handles error recovery (parity, block control check); initializes the LPH and channel control program; processes interrupts, timeouts, and messages; and handles protocol acknowledgment/negative acknowledgment.
- *Poller* – Used only for poll and select protocols. Queues poll requests, requests the LPH to poll a terminal, and dequeues the request when the LPH has received data from the terminal.
- *MLCP driver* – Sets up and processes I/O up to an LPH request, and services MLCP interrupts, passing them to the LPH.
- *Channel control program* (CCP) – Handles character processing, inserts and deletes protocol headers and framing characters (surrounding a message). An extension of the LPH, the CCP resides in the MLCP and is independent of the central processor; thus character processing overhead is eliminated from central processing.
- *MLCP macro routines*

For details on communications software functions, line protocol handlers, and the control structures used for communications tasks, see the *Communications Processing* manual.

PROGRAM PREPARATION SOFTWARE

The software in this category allows you to write, compile, link, execute and debug an application program. Each of the program preparation components, except for Debug, is invoked by command described in the *Commands* manual.

- *Editor* – Used to create and update, on disk, a source unit written in one of the provided programming languages. It will edit characters, expressions, or lines of text. The Editor is reentrant and can support multiple users. A description of the Editor directive language is found in the *Program Preparation* manual.
- *Macro Preprocessor* – Required to process an assembly language application source unit containing calls to macro routines. A macro routine consists of a specified sequence of assembly language source statements that you want specialized and included in your source module. The Macro Preprocessor creates another source unit with assembly source code replacing the macro calls. A description of the macro preprocessor language statements is found in the *Assembly Language Reference* manual.
- *Assembler* – Translates assembly source statements of a source unit into text of a relocatable object unit and optionally produces a cross-reference listing indicating symbol usage. The Assembler can process source for two hardware components: the Commercial Instruction Processor (CIP) and the Scientific Instruction Processor (SIP). The Assembler supports coding of user-defined generic instructions to be executed on the Writable Control Store (WCS).
- *FORTRAN Compiler* – Translates FORTRAN source statements of a source unit into text of a relocatable object unit and source listing or optionally, assembly language source statements in a source unit. The language is based on the American National Standards FORTRAN 77 subset. Offered in the language are the Instrument Society of America (ISA) extensions for bit string manipulation and task management, and a Honeywell extension for communications. FORTRAN programs and Entry-Level COBOL programs can call each other. FORTRAN is intended for commercial and scientific application programming. A description of the FORTRAN language statements is found in the *FORTRAN Reference* manual.
- *Entry-Level COBOL Compiler* – Translates source statements of a source unit into text of a relocatable object unit. Entry-Level COBOL programs and FORTRAN programs can call each other. Significant features of Entry-Level COBOL include: file handling for sequential, relative, and indexed files; three-dimensional tables and indexing; CALL/CANCEL capability; DISPLAY and COMP-1 data; full American National Standards

editing; 21 verbs; and communications through file management facilities. For descriptions of the Entry-Level COBOL language statements, refer to the *Entry-Level COBOL Reference* manual.

- *Intermediate COBOL Compiler* – Translates source statements of a source unit into text of a relocatable object unit. Intermediate COBOL supports Entry-Level COBOL features plus additional features as listed in Table 2-1. Descriptions of the Intermediate COBOL language statements are given in the *Intermediate COBOL Reference* manual.
- *RPG Compiler* – Translates RPG source statements of a source unit into a set of object units consisting of a root plus multiple overlays. The compiler also produces a file containing Linker directives; user-written Linker directives are thus unnecessary. When the command processor is invoked to process the statements in this file, it invokes the Linker, and supplies it with Linker directives necessary to create an executable bound unit. The compiler supports an RPG language comparable to that in current industry-wide use. Significant features include: look-ahead, control levels and matching fields on input; table and array processing; forms alignment; and editing, detail, and total time functions on output. The compiler generates Commercial Instruction Processor code. A description of the RPG language is found in the *RPG Reference* manual.
- *Linker* – Combines object units that are the output of a compiler or the Assembler and produces a bound unit for subsequent loading. It resolves external references made between object units being linked. Linker directives can be used to create reentrant bound unit files. A description of Linker directives is found in the *Program Execution and Checkout* manual.
- *Debug* – Used to test programs at the assembly language level. Hexadecimal patches can be made to the program. Debug is invoked as a separate task group within the system. A description of the Debug directives is found in the *Program Execution and Checkout* manual.

TABLE 2-1. INTERMEDIATE COBOL FUNCTIONALITY NOT AVAILABLE IN ENTRY-LEVEL COBOL

Compiler Enhancements

- Reentrant object programs
- Generates code for Commercial Instruction Processor (CIP)
- Supports SLIC code
- Provides SORT subroutine call

Language Enhancements

- COMP (packed decimal) data and COMP-2 (double word binary) data
- COPY
- ELSE and NEXT SENTENCE options in IF statement
- SAME RECORD AREA
- UNTIL option in PERFORM statement
- Identifier option in WRITE statement
- VALUE OF clause
- DATA RECORDS clause
- DATE-COMPILED paragraph expanded
- Relative Key: maximum value, 2,147,483,647; not restricted to USAGE DISPLAY
- Number of file descriptions: increased from 20 to 99
- Nonnumeric literals: maximum length, 120 characters; nongraphic characters allowed; continuation of literal permited

UTILITY SOFTWARE

A comprehensive set of utility programs is available to support file management and program development. All utility programs listed below are invoked by commands except for Memory Dump (MDUMP) and Dump MLCP (DUMCP). The usage of the utility programs is described in the *Commands* manual unless otherwise indicated.

- *Compare* – Compares two volumes, files or portions of files for equality, and lists the discrepancies.
- *Copy* – Copies a file or volume. Permits the creation of backup copies of files or volumes, either on tape or on a disk device.
- *Create File* – Creates the specified disk file.
- *Create Volume* – Creates or modifies a volume. Formats and labels a disk or tape volume, creates disk bootstrap records, or renames a disk volume.
- *Dump Edit* (DPEDIT) – Produces an edited logical or physical dump image of memory, or edits and prints out a disk file containing a dump of main memory that was obtained through the MDUMP bootstrap record. (Described in the *Program Execution and Checkout* manual.)
- *Dump Memory* (MDUMP) – Dumps the contents of memory to a disk file when a program aborts or halts, by using the bootstrap record MDUMP on a specially created disk volume. The Dump Edit utility is then used to print the dump. (Described in the *Program Execution and Checkout* manual.)
- *Dump MLCP* – Dumps contents of all or part of Multiline Communications Processor (MLCP) memory. (Described in the *Communications Processing* manual.)
- *File Change* – Changes the contents of a disk sector or control interval.
- *File Dump* – Performs both logical and physical dumps from disk or 9-track magnetic tape; performs physical dump only from 7-track tape; output in both alphabetic and hexadecimal notation.
- *Import/Export PAM File* – Converts members of GCOS/BES partitioned files to and from GCOS 6 variable sequential files; used to transport programs between BES and GCOS 6.
- *ISL Configurator* – Reads ISL (Intersystem Link) directives from a user input file and generates an ISL loader.
- *List Creation Date* – Lists creation dates of files in a directory.
- *List Names* – Lists the file and/or directory entries contained within the specified directory.
- *Patch* – Applies hexadecimal patches to an object unit or bound unit. Provides a facility for program correction without recompilation or reassembling. (Described in the *Program Execution and Checkout* manual.)
- *Print* – Prints the contents of the indicated file on a printer, with vertical spacing control
- *Rename File* – Assigns a new name to an existing file or directory.
- *Reset Map* – Lists the number of logical sectors available for allocation on a disk volume.
- *Transmit File* – Supports file transmission between the Level 6 system and other Level 6 processors, or between the Level 6 and any of the following host processors: Level 62, 64 or 66; Series 200/2000; or non-Honeywell systems that use the BSC 2780 protocol. Three utility programs, described in Section 7, provide the file transmission capability.

The Sort/Merge utility capabilities are described below.

Sort/Merge

Sort and Merge are invoked by separate commands. Sort may also be called from a COBOL, FORTRAN, or assembly language program. The Sort program arranges records of a file in an order based on the values of user-specified record key fields. Merge combines the records of up to six sequentially ordered input files on the basis of record key values. Up to 16 key fields can be specified, with values to be arranged in ascending or descending order according to the ASCII collating sequence. The data type of a key field can be character string, signed binary, packed decimal, or signed/unsigned unpacked decimal. Sort/Merge options include record selection, redefinition or rearrangement of record contents, and deletion of duplicate records. See the *Sort/Merge* manual for a detailed description of these capabilities.

RUN-TIME ROUTINES

Run-Time I/O Routines

The FORTRAN run-time I/O routines provide for data transfer, peripheral or communications device manipulation, and the processing of data as specified in FORTRAN FORMAT statements. These routines use the file system to accomplish open, close, and position file functions, and to read and write formatted and unformatted records. They contain data conversion routines to edit integer, real, logical, and character data for formatted input and output. Only those routines required by a particular FORTRAN program are linked to form the bound unit.

The COBOL run-time I/O routine provides a logical I/O interface for the transfer and processing of data at program execution time. The routine is linked with the program's object unit, and uses the file system to open, close, and position files and to read and write records to peripheral or communications devices. Separate run-time routines are provided for Entry-Level and Intermediate COBOL.

The FORTRAN and COBOL routines produce diagnostic messages to inform the programmer of inappropriate or inconsistent input/output statements.

FORTRAN Run-Time Routines

The software includes FORTRAN mathematical and bit string manipulation routines. These intrinsic functions are available in object module format, linked on an as-needed basis to perform a variety of operations on behalf of a FORTRAN program. Optionally, they can be loaded during configuration as an operating system extension available to all online applications. Some of the operations performed by these routines are:

- o Date and time subroutines
- o Converting to and from integer and real values
- o Truncation
- o Determining the nearest whole number
- o Transferring a sign
- o Choosing: the largest value; the smallest value
- o Finding: the length of a character entity; the square root; the natural logarithm; the common logarithm
- o Computing selected plane and spherical trigonometric functions
- o Performing bit string manipulation operations on integer data: inclusive OR, exclusive OR, products, complement, shift, clear or set a bit, and test a bit value.

FORTRAN routines are available to implement the management of tasks. Functions are provided to:

- o Initiate a task after a designated period of time
- o Suspend a task

Communications programs are provided with two routines, ZFSTIN and ZFSTOT, to test the status of the system buffer prior to issuing a READ or WRITE. Depending on the return status, the FORTRAN program can loop on the test, place itself in the wait state, continue other processing, or issue a READ or WRITE and stall if the I/O buffer is busy.

See the *FORTRAN Reference* manual for details about these routines.

HARDWARE SIMULATORS

The SSIP and DSIP (single-and double-precision scientific instruction processor) provide software simulation of floating-point instructions (add, subtract, multiply, divide, compare, load, store, swap, and negate) that are generated by the FORTRAN Compiler or the Assembler.

The Scientific Branch Simulator provides software simulation of floating-point branch instructions (branch on bit settings of scientific indicator register or scientific accumulator values).

The CIP (commercial instruction processor) simulator provides software simulation of CIP instructions (commercially oriented calculations and operations) that are generated by the Intermediate COBOL Compiler, RPG Compiler, or Assembler.

CONFIGURATION LOAD MANAGER

The Configuration Load Manager (CLM) accepts configuration directives from either a Honeywell-supplied input file or a user-generated input file (CLM_USER) to perform system configuration. Configuration directives are available to:

- o Define system variables (e.g., real-time clock, scientific and commercial processors)
- o Describe peripheral devices and their characteristics
- o Define system, batch, and one or more online memory pools
- o Identify system software and application-specific bound units that are to be permanently resident in the system area of memory
- o Define the communications environment of the operating system

Configuration procedures are summarized in Section 4. A complete description of configuration directives appears in the *System Building* manual.

BES-MOD 400 COMPATIBILITY

The GCOS 6 MOD 400 Monitor and I/O system services are a superset of the GCOS/BES online Executive system services. However, differences exist in:

- o Assembly language programs containing calls to the BES Executive
- o BES object modules that must be imported to execute on MOD 400
- o BES COBOL programs that require the BES COBOL and run-time routines for MOD 400 execution
- o Configuration commands

Appendix A of this manual describes the procedures to be used to convert and execute BES programs under MOD 400.

SECTION 3
FILE SYSTEM

FILE AND PATHNAME CONCEPTS

The operating system controls the definition, description, and visibility of entities in the file system. It contains a set of service routines which provide the capability of creating, deleting, retrieving, and modifying entries in the file system.

The file system is represented by a tree-structured hierarchy. The basic elements of this hierarchy are called *files*. Some files are a special type known as *directories*; other files comprise aggregates of data. In the following discussion these two types of files will be differentiated by use of the terms *directories* and *files*, respectively.

Directories and files are referred to by supplying the File System software with a *pathname*, which is an ASCII character string that uniquely identifies every element within the file system.

Directories

A directory is a file that contains information about other files, such as physical and logical attributes of the files and attributes of the peripheral devices upon which they reside. Files whose attributes are described in the directory are said to be immediately contained in, or subordinate to, the directory. They may themselves be directories, or they may be data files.

At the base of each tree structure is a directory known as the root directory. This is the name of the directory that ultimately contains every element that resides within it either immediately or indirectly subordinate to it. The root directory name is the same as the volume identifier of the volume on which it resides; that is, no volume can contain more than one tree structure. However, there may be multiple tree structures accessible to the File System software at any given time, depending on how many volumes are mounted. When the system is informed that a volume has been mounted, its volume identifier, and hence its root directory name, is entered in a device table. All references to files and directories begin, either explicitly or implicitly, with a root directory name, and therefore every file that is mounted is accessible to the File System software.

A magnetic tape file is considered to be a simple tree structure with a root directory (the volume label) and a single file.

When a volume is first created, it contains only a root directory. The user can create, within this directory, any additional directories required to satisfy the needs of his installation. Consider, for example, a volume that is to contain data used by two application projects, each of which has several people associated with it. Each of these people has one or more files of interest to him. The volume has been initialized and contains a root directory name. Two directories can be created, subordinate to the root directory, each identified by the project name. Then subordinate to these directories, a directory can be created for each person associated with each project. This directory level is the one at which each person will normally operate. His data files are all contained within his personal directory, either immediately subordinate to it or subordinate to subdirectories, which he may create to reflect his particular needs.

When the need for a directory no longer exists, the directory can be deleted from the file system, making the space it occupied, as well as the space occupied by its attributes in the immediately superior directory, available for reuse. A directory must be empty before it can be deleted; all directories and files subordinate to the one to be deleted must have been previously deleted by explicit commands.

Files

A file is defined as any unit of storage external to the central processor that is capable of supplying data to, and/or receiving data from, a task. Under this broad definition, a file can be simply a peripheral device, such as a printer, card reader, or terminal device, or it can be an aggregate of data stored within a directory structure, such as that described in the previous paragraphs, on a magnetic storage device. The conventions used to refer to any of these types of files are essentially the same; only the complexity of the file's unique identifier varies with the type and location of the file.

A file is always the endpoint of any branch in a tree structure. That is, it contains no information that the system interprets as attributes of subordinate directories or files. It is thus the basic, or lowest level, structural unit that can be referred to through the File System software.

Files can be created and deleted either explicitly or implicitly. Explicit creation and deletion are done through the use of execution control commands issued by the user. Implicitly created files, used mostly as temporary work files, result from the use of many of the program development components, such as the Editor, Assembler, Linker, and some utility components. These files are deleted upon normal termination of these components; their creation, deletion, and existence are largely invisible to the user. However, some files are implicitly created but not deleted; chief among these are the files produced by the Assembler, the compilers, and the Linker for subsequent listing by a utility program. These must be explicitly deleted after they have served their purpose. The names by which they are known, and by which they must be deleted, are given in the various detailed descriptions of these components.

Pathnames

The discussion of file concepts thus far has presented the concepts of directories and files from the point of view of their existence and function only. The following material describes the way in which these entities are named and how these names are used to construct unique identifiers by which each such entity may be referenced.

Naming Conventions

Each directory or file name in the file system can consist of ASCII characters from the following sets:

- Uppercase alphabetic (A through Z)
- Numeric (0 through 9)
- The underscore (_)
- The period (.)
- The dollar sign ($)

The first character of any name must be either an alphabetic or the dollar sign ($). The underscore character can be used to join two or more words that are to be interpreted as a single name (e.g., DATE_TIME). The period character followed by one or more alphabetic or numeric characters is normally interpreted as a suffix to a file name. This convention is followed, for example, by a compiler when it generates a file that is to be subsequently listed; the compiler identifies this file by creating a name of the form "FILE.L.".

The name of a root directory or a volume identifier can consist of from one to six characters. The names of other directories and files can comprise from 1 to 12 characters. The length of a file name must be such that any system-supplied suffix does not result in a name of more than 12 characters.

Pathname Construction

The access path to any file system entity (directory or file) begins with a root directory name and proceeds through zero or more subdirectory levels to the desired entity. The series of directory names (and a file name if a file is the target entity) is known as the entity's *pathname.* The total length of any pathname, including all hierarchial symbols,

cannot exceed 58 characters, except that a working directory pathname cannot exceed 44 characters.

In the construction of a pathname, certain symbols are used to indicate the hierarchial relationship between the pathname's elements. These symbols and their meanings are shown below.

- Circumflex (^) – Used exclusively to identify the name of the root directory. It precedes the root directory name, thus: ^VOL011.
- Greater than (>) – Indicates movement in the hierarchy away from the root directory. The symbol is used to connect two directory names or a directory name and a file name; it can also be the first character of a pathname, in which case it is immediately subordinate to the root directory of the system volume. Each occurrence of the symbol denotes a change in the directory level; the name to the right of the symbol is immediately subordinate to the name on the left. Reading a pathname from left to right thus indicates movement through the tree structure in a direction *away from* the root directory. If the root directory ^VOL011 contains a directory name DIR1, then the pathname of DIR1 is

  ^VOL011>DIR1

  If the directory named DIR1 in turn contains a file named FILEA, then the pathname of FILEA is

  ^VOL011>DIR1>FILEA

- Less than (<) – Used at the beginning of a pathname to indicate movement through the tree structure in a direction *toward* the root directory. Consecutive symbols can be used to indicate changes of more than one level; each occurrence represents a one-level change. When followed by elements of a relative pathname, those elements represent changes of direction *away* from the root directory. One or more of these symbols may only precede a relative pathname.
- ASCII "space" character – Used to indicate the end of a pathname. When represented in memory, a pathname must end with a space character.

The last element in a pathname is the name of the entity upon which action is to be taken. This element can be either a directory name or a file name, depending on the function to be performed. In the CREATE DIRECTORY command, for example, a pathname specifies the name of a directory to be created. The last element of this pathname is interpreted by the command as a directory name; any names preceding the final name are names of superior directories leading to it. An analogous situation occurs in the CREATE FILE command, except that in this case the final pathname element is the name of a file to be created.

The pathnames described to this point can be termed *full* pathnames, in that they contain all necessary elements to describe a unique access path to a file system entity, regardless of the type and location of the device on which it resides. The file access system uses this form in referring to a directory or file. However, it is frequently unnecessary for the user to specify all of these elements; the system can supply some of them under certain conditions; i.e., when the missing elements are known to the system and the abbreviated pathnames are used in the appropriate context. An understanding of these conditions and contexts requires an understanding of absolute and relative pathnames and the concept of the working directory. These subjects are described in the following paragraphs.

Absolute Pathnames

An absolute pathname is one that begins with a circumflex (^) or a greater-than symbol (>). A pathname that begins with a circumflex is called a full pathname. This form is used to reference directories and files that reside on a device other than that on which the system

volume (the volume from which the system was initialized) is mounted. When an absolute pathname begins with a greater-than symbol, the first element named in the pathname is assumed to be immediately subordinate to the system volume root directory. Thus, if the system volume name is SYS01 and the pathname given is >DIR1>FILEA, the full pathname becomes ^SYS01>DIR1>FILEA.

Another volume, USER1, can also contain a >DIR1>FILEA access path and can be known to the system; the two access paths are made unique by requiring that the root directory be specified when referring to the second volume. The full pathname of this file on the second volume is thus ^USER1>DIR1>FILEA.

Relative Pathname and Working Directory

A relative pathname is one that does *not* begin with the circumflex or greater-than symbol. For a relative pathname that does not begin with a less-than symbol, the first (or only) name in the pathname identifies a directory or file that is immediately subordinate to a directory known as the *working directory*. The working directory is the user's current position in the file system hierarchy.

A *simple* name is a special case of the relative pathname. It consists of only one element: the name of the desired entry in the working directory.

Working Directory

The initial setting of the working directory is derived from values in the -WD argument of the EGR (enter group request), EBR (enter batch request), or SG (spawn group) command, or the -HD argument of the LOGIN command. (Refer to the *Commands* manual for details.) This directory can be changed by system service macro calls and commands.

A relative pathname can consist of one or more elements. If a relative pathname contains more than one element, each element except the last is a directory name, the first immediately subordinate to the current working directory level, the second immediately subordinate to the first, and so on. The last or only element can be either a directory name or a file name, depending on the function being performed, as described previously.

In some cases, it may be necessary for a user to refer to a file contained in a directory subordinate at some level to the same directory as that to which his own working directory is subordinate. He has two alternative ways of making this reference; he can use an absolute pathname, or he may use a special form of relative pathname that beings with a less-than (<) symbol.

Figure 3-1 shows some relative pathnames and the full pathnames they represent when the working directory pathname is:

>UDD>PROJ1>USERA

Device Pathnames

Reference to any device is through the Symbolic Peripheral Device (SPD) directory, which is subordinate to the system root.

Device Files (Other Than Disk And Tape) – The general form of a device file pathname is:

>SPD>dev_name

where dev_name is the symbolic name defined for the card reader, punch, printer, or terminal device during system building.

Tape Files – The general form of a tape file (device) pathname is:

>SPD>dev_name[>volid[>filename]]

where dev_name is the symbolic name defined for the tape device during system building, volid is the name of the tape volume, and filename is the name of the file on the volume. Tape devices are always reserved for exclusive use; i.e., the reserving task group has read and write access, but other users are not allowed to share the file.

Disk Device Files – The general form of a disk device-level access pathname is:

>SPD>dev_name[>volid]

where dev_name is the symbolic name defined for the disk device during system building, and volid is the name of the disk volume.

This pathname format is used only when access to the entire volume is required, e.g., during a volume copy or volume dump.

If the volid is not supplied, reservation of the disk is exclusive; i.e., the reserving task group has read and write access, but other users are not allowed to share the file. This pathname form is used when creating a new volume.

Relative Pathname	*Full Pathname*
DELTA	^SYS01>UDD>PROJ1>USERA>DELTA
OLD>DELTA	^SYS01>UDD>PROJ1>USERA>OLD>DELTA
<USERB>ALPHA	^SYS01>UDD>PROJ1>USERB>ALPHA
<<PROJ2>USERA>DELTA	^SYS01>UDD>PROJ2>USERA>DELTA
<	^SYS01>UDD>PROJ1

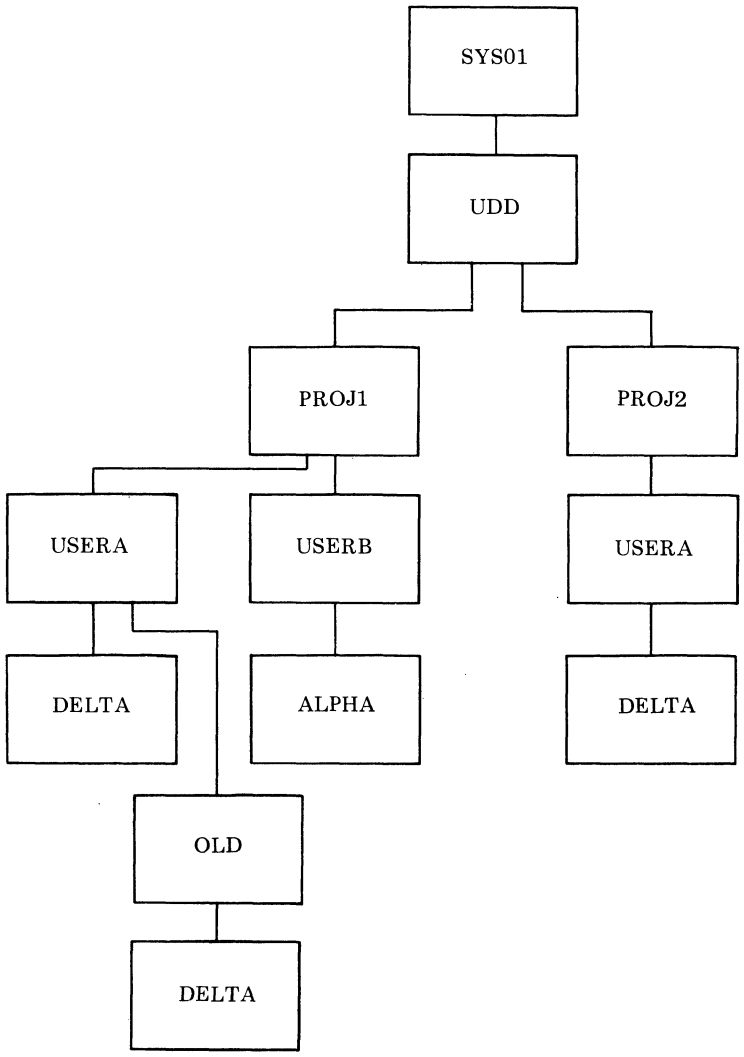


Figure 3-1. Sample Pathnames

If the volid is specified, reservation is read/share; i.e., the reserving task group has read access only, other users may read and write. This pathname form is used when dumping selected portions of a volume without regard to the hierarchial file system tree structure.

The following are examples of device pathnames:

Peripheral Device	*Pathname*
Line printer	>SPD>LPT01
Exclusive tape volume	>SPD>MT902>VOL3
File on exclusive tape volume	>SPD>MT902>VOL3>FILEA
Exclusive diskette	>SPD>DSK02
Nonexclusive disk volume	>SPD>RCD01>V23X

Special Pathname Conventions

Star names and equal names are special pathname conventions available for use with certain utility programs. They provide shorthand methods of specifying a related set of entry names to commands.

Star Convention

A star name is an entry name that identifies a group of entries within a single directory. It is composed of one or more nonnull components, one of which consists of an asterisk (star), and can contain up to 12 ASCII characters, none of which can be the greater than (>), less than (<), or circumflex (^) characters.

A star name can be used only as the final entry name of an input pathname for the following commands:

- o COPY
- o COMPARE
- o LIST NAMES
- o LIST CREATION DATE

These commands perform their function for each entry identified by the star name.

A star name identifies all directory entries having an entry name that matches the star name. A special type of matching is performed in which components of a star name that do not contain an asterisk are compared with corresponding components of an entry name, while other entry name components are ignored. Entries identified by a star name all have similar names in that they are all determined by the star name template. For example, the star name *.IN.A refers to all three-component entry names ending in .IN.A (Macro Preprocessor include files) in the working directory.

An asterisk may appear in any component position of an entry name; each asterisk is treated as a special character. Each asterisk character used in a star name designates any number of characters (including none) appearing in the corresponding component position of the entry name. A double asterisk (**) may be used to indicate any number of contiguous component positions within an entry name. One or more single asterisks may appear in a star name; only one double asterisk component may be used. For example, the star name *.MY_PROG.** identifies all multiple-component file names, the second component of which is MY_PROG, in the working directory.

A question mark character (?) may be used within a star name to match any character position that appears in the corresponding component and character position within that component of the entry name. Multiple question marks may be specified, each representing exactly one character position. For example, the star name *.MY_?????.* identifies all three-component file names, the second component of which is an eight-character name beginning with MY_, in the working directory.

For complete details concerning the star convention, refer to the *Commands* manual.

Equal Convention

Two commands that accept pairs of pathnames as their parameters (i.e., COPY and COMPARE) allow the final entry name of the first pathname (input) to be a star name and

the final name of the second pathname (output) to be an equal name. An equal sign as a component of the output pathname means that the character string from the corresponding component of the input pathname is to be substituted for the equal sign. Use of this convention allows the user to copy or compare multiple input files without specifying complete individual names. Output file pathnames are determined by the equal name convention after the input file pathnames containing star names have been resolved.

The equal names convention provides a powerful mechanism for mapping certain character strings from the first pathname into the second pathname of the pair. Such a mechanism helps to reduce typing required to specify the second pathname, and it can be essential for mapping character strings from the entry names identified by the star name into the equal name, because these character strings are not known at the time the command is issued.

Under the equal convention, the mapping of character strings from the star name into the equal name is performed according to rules for constructing and interpreting equal names. An equal name is an entry name composed of one or more nonnull components, one of which consists of an equal sign, and can contain up to 12 ASCII characters, none of which can be the greater than (>) or less than (<) characters. An equal sign can appear in any component position of an entry name; each is treated as a special character. The equal sign represents the corresponding component of an entry name identified by the star name. An error occurs if the corresponding component does not exist. A double equal sign (==) component can be used to represent all components of entry names identified by the star name that have no other corresponding components in the equal name.

The percent sign (%) functions in the same manner as the question mark in a star name. The percent sign can be used within an equal name to match any character that appears in the corresponding component and character position within that component of the entry name identified by the star name. An error occurs if the corresponding character does not exist or if an equal sign appears in a component that also contains a percent sign. Multiple percent signs can be specified, each one representing exactly one character positon.

For example, the command

```
COPY RANDOM.DATA_BASE SORTED.=
```

creates a duplicate copy of the input file in the working directory, but assigns the name SORTED.DATA_BASE to the duplicate file.

Likewise, the star convention can be used to address all files with a specific component name in the working directory. The command

```
COPY *.DATA_BASE =.DATA
```

copies all two-component entry names with DATA_BASE as the second component and assigns the name DATA as the second component of the newly created files.

DATA FILE ORGANIZATIONS AND ACCESS

The operating system supports four disk file organizations and one magnetic tape file organization. Descriptions of data file organizations and their file formats are found in the *Data File Organizations and Formats* manual.

Data File Organizations

The file organizations for disk are:

Sequential – Records are accessed in consecutive order. Variable-length records are handled in variable-length formats. A record can be updated (i.e., rewritten) or appended to the file. If a record is deleted, the position it occupied cannot be reused.

Relative – Records are accessed sequentially or directly by their record position relative to the beginning of the file. Variable-length records are handled in fixed-length formats. A record can be updated (i.e., rewritten), deleted, or appended to the file. If a record is deleted, the position it occupied can be used for a new record.

Indexed – Records are logically ordered by record key value. Records can be accessed sequentially in key sequence or directly by key value. Variable-length records are handled in variable-length formats. A record can be updated, deleted, or inserted in key sequence into available free space. When no space is available in key sequence for an inserted record, the record is placed in an overflow area.
Fixed Relative – Records are accessed sequentially or directly by their record position relative to the beginning of the file. Only fixed-length records are supported. A fixed-relative file can be restricted to nondeletable records. A record can be updated or appended to the file; if deletable records are allowed, the deleted record position may be used for new record data. The fixed-relative file organization is used in applications where the file is to be used on or is from a BES1 or BES2 Executive system. A fixed-relative file is incompatible with file organizations on other GCOS systems.

The only file organization supported for magnetic tape is *Sequential.* A tape file can contain fixed- or variable-length records. A record can be appended to the file, but it cannot be updated or deleted.

Data File Access

The languages COBOL, FORTRAN, and RPG use logical file organizations as described above. Refer to the language reference manuals for the relationship between a language's logical file and the system's physical file organization. Each language provides statements for manipulating files. Files can be manipulated by assembly language programs through File and Data Management macro calls to the Monitor or through the physical I/O drivers; both methods are described in the *System Service Macro Calls* manual.

File Concurrency

Concurrent read or write use of a file is established by the task group that reserves the file. Concurrency has two aspects: (1) it establishes how tasks in the reserving task group intend to access the file, and (2) it establishes what the reserving task group allows other task groups to do with a file. If the file is already reserved, a task group's concurrency request will be denied when its intended access conflicts with the access permitted by another task group. The concurrency request will also be denied if what it allows others to do conflicts with the access already established by another task group. For example, if a task group reserves the file exclusively, other task groups are denied access. Of if a task group permits read only access but does not permit write access, other readers are allowed but writers are denied access.

Concurrency is controlled through the GET command or through the $GTFIL system service macro call. The possible combinations of access intended for the reserving task group and the sharability permitted other task groups are given in Table 3-1.

TABLE 3-1. DISK FILE CONCURRENCY CONTROL

Reserving Task Group	Other Task Groups
Read only	Read only (Read share)
	Read or Write (Read/write share)
Read or Write	No read, no write (Exclusive use)
	Read only (Read share)
	Read or Write (Read/write share)

System File Concurrency

Compiler generated programs, commands, Sort, and other system software always request exclusive concurrency for files that they reserve for a user. The operator terminal must be reserved with read/write shared concurrency to allow concurrent access by many task groups. For this reason, the command argument -COUT specifying the list output file cannot be the operator terminal. If the command-in and user-in files are on

disk, they are reserved with read-only shared concurrency; if assigned to a user terminal, they are reserved with exclusive concurrency. The user-out and command-out files are always reserved for exclusive use.

Record Locking (Shared File Protection)

The record locking facility is an optional file access feature that provides protection of and controls contention for records within shared disk files. The first user to access (read or write) a record locks the record, making it inaccessible to other concurrent users of the file until the task group causing the lock releases the record. Any contention for the locked record causes the system to note the interference and to issue an error return code.

The capability to perform record locking is a configuration option that the user must specify during system initialization. Locking is performed on a file by a specific request when the file is reserved; this is accomplished by a $GTFIL system macro call or by a GET command that uses the -LOCK argument. Locking is performed on a control interval basis on either a READ or WRITE operation. When the user accesses an unlocked record in a shared file, the control interval containing that record is marked for exclusive use. Other users sharing that file are denied access to that record and to any other record contained in the control interval until the record is unlocked. Subsequent attempts by other users to access a locked record cause interference, in which case access is refused and the user creating the lock is notified. Locked records are released upon explicit user request ($ULREC), when the file is closed, or when the task group terminates.

The first reserver of a file establishes the pattern for all other users of the file. If the user who first reserves the file does not request locking, no subsequent reservation of the file can request locking. Conversely, if the user who first reserves the file requests locking, all subsequent reservations of the file must also request locking. A user, however, can always read records whether or not they are locked by another user by specifying read-only access with read/write sharing by others. The integrity of the data he reads is not guaranteed. (Concurrency control is described under the GET command in the *Commands* manual and under the $GTFIL macro call in the *System Service Macro Calls* manual.)

FILE SYSTEM BUFFERED OPERATIONS

A buffer is a storage area used to compensate for a difference in rate of flow of data, or time of occurrence of events, during transmission of data from one device to another. As used in I/O programming, the term buffer refers to an I/O area in systems that provide the possibility of I/O overlap. Buffering is the process of allocating and scheduling the use of buffers. In sequential data processing, for example, overlap of input operations and processing can be achieved by anticipatory buffering; i.e., the next block is read into memory before it is needed. The program can then process records from block n while block n+1 is being read into memory.

This system supports two types of buffered operations: one for unit record and terminal devices, the other for disk and magnetic tape devices.

Unit Record and Terminal Buffered Operations

All printers and *most* interactive terminals are provided with one File System buffer. To provide a system buffer for the card reader or terminals configured as file types KDL, TDH, or TDL, the B parameter in the CLM DEVICE directive must be specified. The operator terminal (system LRN 0) cannot be buffered. By providing a File System buffer, asynchronous I/O can be done; i.e., application code can execute in parallel with I/O transfers.

All terminals (except the operator's) and printers, except file types KDL, TDH, and TDL, have tabbing capability through software that converts the tab into spaces. Default tabulation stops are set at position 11 and every tenth position thereafter for the line length of the device.

Asynchronous I/O operates in two different ways, depending on whether data is obtained from a device (reading) or transferred to a device (writing).

Buffered Read Operations

An application task issues a logical READ to a File System buffered device. If the buffer is full from a prior anticipatory read, the data in the buffer is transferred into the application task's area; then a physical I/O transfer into the system buffer (an anticipatory read) is performed in parallel with continued task execution. If the buffer is not full, task execution stalls until the anticipatory read is completed.

The timing of the initial anticipatory read performed for the card reader is different from that of the interactive terminals; afterwards it is the same. An application task issues an OPEN call to the card reader. Immediately after the OPEN is complete the FILE System performs an asynchronous anticipatory read into the system buffer while the application continues execution. All OPEN calls are synchronous.

For interactive terminals, immediately after the OPEN is complete an asynchronous physical connect is performed while the application continues execution. Assembly or FORTRAN applications can check the status of the OPEN to see if a READ can be issued without stalling application execution. File System issues an asynchronous anticipatory physical read when a status check after the physical connect is complete. The file status remains busy until the physical read is done and the system buffer is full. At this point, the file status is "not busy" (i.e., the anticipatory read is successfully completed), and the application can issue a READ with the assurance of receiving data immediately. If at any point after the OPEN is issued the assembly or FORTRAN application issues a READ before the physical connect and anticipatory read have been completed, the READ is synchronous and further central processor execution is stalled on this application until the anticipatory read is complete. To avoid status check looping to test the input buffer status or stalling on a READ, both assembly and FORTRAN applications can put themselves into the wait state, thus making the central processor available for lower priority tasks. After the OPEN, a COBOL application must issue READ requests. The COBOL application will be put in wait state if it is executing its I/O statements in synchronous mode. Otherwise, the COBOL run-time package performs the status checks and returns a 9I status until successful completion. The COBOL program can either loop on the READ or continue other processing.

The anticipatory read allows an application to control input from more than one interactive terminal, each of which represents a data entry terminal. By testing the status of the system buffer before a READ (FORTRAN, assembly) or by checking for the 9I status return after a READ (COBOL) even if a terminal operator is not present at the time of the READ request, the application will not be stalled and it can continue to poll other terminals.

Buffered Write Operations

A buffered write operation to a device 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 call and no distinction is made between interactive and noninteractive file types. Each WRITE call is completed by moving data from the application buffer to the File System's buffer (performing any detabbing, if requested), initiating the transfer, and returning control to the application program. If the program performs a second WRITE while the system buffer is still in use for a previous transfer, the application is stalled until the buffer is available and new data moved into it again. The application can avoid stalling execution by checking the status of the system buffer before issuing a WRITE to an interactive terminal to see if, in a special mode, it is still in use or not (FORTRAN, assembly) or by testing for the 9I status return after the WRITE (COBOL, for interactive devices only).

If a WRITE call is issued while data is being entered (because of a read) into the system buffer, the read is allowed to complete and input data is saved in the system buffer, a synchronous write is reissued by File System, and output data is transferred directly from the application buffer. However, tab characters are not expanded into spaces by software.

Special considerations for buffered write operations arise because, if a physical I/O error occurs while data is being transferred from the system buffer to the device, the application program must be aware that the error occurred on the previous write operation. 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.

Disk and Magnetic Tape Buffered Operations

An assembly language application can request buffers through the system service macro call to get a file for reservation. If File System needs buffers for blocking or unblocking, it provides either the number of buffers specified or, by default, one buffer. If no buffers are needed, none are provided, even if specified. Each buffer contains a disk control interval (CI) or magnetic tape block. When an application program issues a READ and the desired record is not in any buffer, the next empty available buffer is filled with the CI or block containing the record; when all buffers are filled, an active buffer is selected for the next different CI or block based on a least recent usage algorithm.

I/O assembly language macro calls for disk or magnetic tape operations are synchronous, and the application stalls until the I/O operation is completed. Asynchronous I/O can be obtained through File System Storage Management macro calls and, optionally, through physical I/O (PIO).

SPOOLING TECHNIQUE

SPOOLing (Simultaneous Peripheral Operations On-Line) is an I/O technique that allows the reading and writing of input and output streams on auxiliary storage devices, concurrently with job execution, in a format convenient for later processing or output operations.

The following method can be used to spool printer output to one disk device from both online and batch task groups.

1. Define an online task group whose function is to produce print output files, for example:

   CG DP 3 -EFN PR -POOL AB

   where PR is the print utility and the other arguments are arbitrary.
2. Put the printer output files on shareable disk files by setting the "user-out" file to a disk file, which will be created by the FO command if it does not exist, and/or by using the "-COUT path" option on the assembler and compilers. The -COUT option puts bulk output on a specified file and also creates the file if it does not exist.
3. Later when a rpint file is to be printed, issue the following command:

   EGR DP -OUT >SPD>LPTxx -ARG fullpath [optional PR control]

   where "fullpath" is the full pathname of the file to be printed. Optional PR control arguments could include -RL, which will release (delete) the print file if normal termination occurs. The LPTxx is the printer used by the print utility that prints the spooled data. This example assumes that there is no concurrency conflict for printer use at the time of printing.

SECTION 4

SYSTEM ACCESS

SYSTEM CONFIGURATION AND ENVIRONMENT DEFINITION

At larger installations a system programmer might design the configuration files and the possibly different operating environments to be used at the installation. The daily startup would be done by an operator. At smaller installations, especially those where programmers run dedicated applications, each programmer might do the configuration and startup for his application.

Creation of a usable system consists of a two-step procedure:

- o Bootstrap a Honeywell-supplied system startup routine that provides a limited operating environment for building the files used in the second step
- o Specialize the system startup procedure by configuring a system to correspond to the installed hardware and by defining the environment in which to prepare and execute applications programs

The bootstrap operation simply consists of turning on the power supply to the hardware, mounting the cartridge disk or diskettes containing the MOD 400 operating system software, and pressing several control panel keys including bootstrap load to execute a standard bootstrap routine. The bootstrap operation includes the initial configuration and startup operations; procedures are executed (1) to configure a limited system consisting of cartridge disks storage modules, or diskettes, and operator terminal, and (2) to provide a one-user online environment that can be used to specialize system startup, perform program preparation, or perform application program execution.

In the provided user environment, the user can employ the Editor to create two files that specialize system startup:

- o CLM_USER – Contains configuration directives, which when executed, will configure a system to correspond to the actual installation hardware
- o START_UP.EC – Contains operator commands, which when executed, will define the installation-specific operating environment consisting of task groups

When these files have been created, the system is again bootstrapped. However, this time directives in the CLM_USER file control the configuration, and operator commands in the START_UP.EC file define the operating environment. (Refer to the *System Building* manual for complete details.)

Configuration directives are available to perform the following functions:

- o Describe available central processing unit options, such as the real-time clock, scientific processor, additional overlay areas, and trap save areas.
- o Describe peripheral and communications devices and their characteristics.
- o Specify the memory pools that partition memory. The system and each user task group, under which applications execute, must be associated with a single memory pool. System and user memory pools are discrete.
- o Indicate which operating system overlay areas should be permanently resident.
- o Indicate that an application-specific bound unit should be permanently resident and be part of the operating system.

The START_UP.EC file is described later in this section.

ACCESSING THE SYSTEM

Ways to Access the System

An installation can simultaneously support several ways to access a system, so a user must determine the access available at a terminal. For simplicity of discussion, three types of access will be described. Access can be (1) through a LOGIN command, (2) through operator control, or (3) through a user's own applications design.

Logging In

The login function allows a user, without operator intervention, to activate an application from any one of the designated terminals. The login function can be activated by the operator after configuration is complete as described in the *Operator's Guide.* Depending on the capability designed and configured for an installation, a user can login in one of three ways. A user can:

1. Type in a LOGIN command as described in the *Commands* manual.
2. Type in the abbreviation of a specific LOGIN command line. A file contains, for each abbreviation, an image of the LOGIN command line that can either be used at all login terminals or that might be restricted to designated ones.
3. Turn on the terminal and be logged in through a direct login. Direct login is useful for transaction processing applications where the user wants to interact with the application and not the operating system, e.g., an application to provide, on request from the designated direct login terminals, the current inventory of a product.

Operator Assigned Access

The operator or another user must activate the application that is to be run and also designate the terminal that is to be used to input commands or user input required by the program executing the command. Terminals that are used for logging in cannot be assigned by the operator or another user. An installation can have a mixture of terminals: some that may be used for logging in and others that may be assigned through the operator or another user.

User Designed Access

A user at an installation that allows use of the system for a single dedicated application, must configure and startup the system, act as operator, determine what the application environment should be, and how to access the system for that application. If an installation has one terminal, it is used both as an operator terminal and user terminal, as described in the *Operator's Guide.*

The Activated Lead Task

When a user successfully gains access to the system, executable code for the lead task (i.e., the controlling task of the application) is loaded and activated. The lead task can be designated to be either the command processor or a user application. When the command processor is the lead task, the user has complete flexibility to control execution by being able to execute any command in the *Commands* manual. When an application is the lead task, the command processor is not part of the task group.

COMMAND ENVIRONMENT

The command environment is that environment in which the user can communicate with the operating system through the use of command lines entered at a terminal or read from a command file. The essential parts of the command environment, from the user's point of view, are the command processor and the command input file (command-in). The command processor is the system software component which reads command lines issued by the user. It interprets them into procedures that load and initiate execution of bound units which fulfill the requests represented by the command lines. The command input file (command-in) is the file from which the command lines are read. It can be a

terminal device, as in the case of an interactive user, or a command file stored on disk or on cards, as in the case of a noninteractive user.

Three other files are involved with, but not limited to, the command environment. These are the user input file (user-in), the user output file (user-out), and the error output file (error-out).

The user-in file is the file from which a command function, during its execution, reads its own input. When a task group request has been processed, and as long as no alternate user-in file is specified as an argument in a subsequent command, the user-in file remains the same as the command-in file. At the termination of a command which names an alternate user-in file, the user-in file reverts to its initial assignment. The directives submitted to the Editor following the entry of the EDITOR command, for example, are submitted through user-in. No specific action is required on the user's part to activate, or to connect to, user-in unless the directives are to be read from a previously-created disk file. The user simply invokes the Editor and begins entering editor directives through the same terminal; the attaching of the terminal to the user-in file is invisible to the user.

The user-out file is the file to which a task group normally writes its output. However, certain system components (compilers, etc.) also write to list files (path.L) or to the output file defined in the -COUT argument. The user-out file is initially established by the -OUT argument of the EBR, EGR, or SG command. (Thus, originally, it is the same device as the error output file device.) It can be reassigned to another device by use of the FO (file out) command or by the use of the $NUOUT (new user out) system service macro call. Such a reassignment remains in effect for the task group until another reassignment occurs. Again using the Editor as an example, any responses from the Editor, such as the printing of a line of the file being edited, are issued through user-out. As in the case of user-in, no special action is required of the user to attach his terminal to the user output file. The only time such action would be required is if the output from the command were to be directed to some device other than the terminal.

Error-out is used by the system to communicate to the user an error condition which may be detected during the interpretation of a command or its subsequent execution. Such a condition could be a missing command argument, reported by the command processor, or a file not found condition, reported by the invoked command. The error output file is the same as the initial user-out file. The user cannot reassign error-out through a command, only through a $EROUT system service macro call.

Subsequent paragraphs in this section describe in detail the functions available to the user at command level.

COMMAND LEVEL

When the system is in a state capable of accepting a command from command-in, it is said to be at command level. The methods whereby command level is achieved and the functions that the system performs while at command level are described in the following paragraphs.

Achieving Command Level

Command level can be achieved in any of several ways. Regardless of the way in which the system arrives at this state, the system indicates that it is at command level by issuing a "ready" prompter message at the user's terminal. (This assumes that the user has not disabled the ready message by issuing a READY_OFF command; if he has, the system still comes to command level but the user is not informed.)

A user is initially at command level when the lead task of a user task group is the command processor.

When executing a command function, command level can be returned to in one of two ways.

- At normal termination of a command function, the system returns to command level and awaits the entry of another command. This command can be some other function that the user desires to execute, or it can be a BYE command, indicating that he has no further work to do and wishes to terminate the current session.

- The user can interrupt the execution of an invoked command by pressing the "break" or "interrupt" key on his terminal. The system then responds with the break message. At this point he can enter the START command to resume processing where it was interrupted, or he can enter a new command as described in the *Commands* manual.

Functions Performed at Command Level

When a command such as COPY, CWD (change working directory), or EGR (enter group request) is read by the command processor, the system spawns a task whose objective is to fulfill the requirements of the command. This action effectively consists of the following steps:

- A task is spawned naming the requested bound unit i.e., command name. Task spawning implies task creation, i.e., the allocation and initialization by the system of any control structures and data areas required for task control.
- The loader is called to load the requested bound unit
- A request for its execution is placed against the created task and the command processor enters the wait state to await completion of the requested task (command). At this point the system leaves command level, which can be returned to only by completion of execution of the command or by pressing the "break" or "interrupt" key on the terminal, as described previously.
- If the command is an EGR, it places a group request against an application task group and then the EGR command terminates. The request is queued if there are other outstanding requests against the application task group from previous EGRs.
- When the command terminates, the spawned task is deleted and a ready message is optionally issued to indicate that the system has returned to command level and can accept further commands.

COMMAND LINE FORMAT

Commands are read and interpreted by the command processor, which executes the lead task in the batch task group, or can execute as the lead task in an online task group. Each command causes a task to be spawned within this task group to perform the requested function (e.g., create a task within an existing group, enter a group request, dump a file). When the execution of a command terminates, control is returned to the command processor, which can then accept another command.

A command line to the processor is a string of up to 127 characters in the form

command-name [arg_1 . . . arg_n]

where command-name is the pathname of the bound unit that performs the command's function. Each subsequent arg entry is an argument whose functions are described in the following sections. A command line cannot be continued onto the next line.

Arguments

An argument of a command is an individual item of data passed to the task of the named command. Some commands require no arguments; others accept one or more arguments as indicated in the syntax of each command description. The types of arguments used are:

- Positional argument – An argument whose position in the command line indicates to which variable the item of data is applied. The argument can occur in a command line immediately after the command name or as the last argument following the control arguments, as in the LIST NAMES command.
- Control argument – A keyword whose value specifies a command option. A keyword is a fixed-form character string preceded by a hyphen (e.g., -ECL). It can be alone, as in -WAIT, or it can be followed by a value, as in -FROM xx.

Except for -ARG or when the last argument of a command line is a positional argument, keywords of control arguments can be entered in any order in the line, following the initial

positional arguments. The keyword -ARG must be the last argument of a SG, EBR, ETR, or ST command line. The arguments following the -ARG are passed to the activated (application) task.

Spaces in Command Lines

Arguments in command lines are separated from each other by spaces. Unless otherwise indicated, a space in a command line syntax represents one or more space characters, or one or more horizontal tab characters, or a combination of these. Spaces can be embedded within an argument by enclosing the argument in single (') or double (") quote character. For example, a file name supplied in an argument should have a trailing space if the argument is bounded by quotes.

Parameters

Arguments are the user-selected items of data passed to a task. In the activated task, which is written in a generalized manner to handle any set of data passed to it, this data is known as parameters. If the activated task expects positional parameters, the order of the command line arguments passed to it must be in the same order as the task's positional parameters.

Protected Strings

Special significance is attached to the following reserved characters:

- o Space (blank)
- o Horizontal tab
- o Double quote (")
- o Single quote (')
- o Semicolon (;) (in assembly language)
- o Ampersand (&)

It is occasionally necessary to use a reserved character without its special meaning. For example, a blank could be used in a command argument. The protected string designators (the double and single quote) are reserved for this purpose. Reserved characters within a protected string (one surrounded by protected string designators) are treated as ordinary characters. Thus, in an argument

```
-ARG "ALPHA 2" ALPHA
```

the space in ALPHA 2 is treated as part of the name.

Another example is the & followed by a numeric in a command-in file. If, for example, &1 is not to be interpreted as a substitutable parameter, it must be written as &'1' or &"1" (not "&1").

Also, since protected string designators themselves are reserved characters, it may be desirable to suppress their special meaning. For this purpose, two adjacent protected string designators of the same type within a protected string of that type are treated as a single occurrence of that character. A protected string designator within a string enclosed by the complementary protected string designator is treated as an ordinary character. Both of the following arguments

```
-ARG "A""B"

-ARG 'A"B'
```

result in the string A"B being passed to a command.

EC FILES

The command processor is able to read commands from a source other than an interactive user terminal. Such a source can be in the form of an EC file. An EC file is one which is con-

structed by the user, e.g., using the Editor, and which is destined to be read by the command processor invoked either by the EC command or when a task group is activated with the command processor as its lead task. The EC file could contain a series of commands which the user executes on a frequent basis, such as commands to execute a set of applications programs that are run at the end of the month to summarize inventory, sales, and accounts receivable.

Startup EC Files

A special application of EC files is their use when activating a task group.

After configuration, i.e., after the CLM_USER file of configuration directives is executed, a user written command file, START_UP.EC, attached to the root directory is executed, if present. It contains operator commands used, for example, to establish an applications environment for that installation.

Also, when a task group is activated whose lead task is the command processor, the command processor will first execute the EC file, working_directory>START_UP.EC, if there is one. This file could contain commands used, for example to execute in sequence, without further user intervention, the tasks of the job.

SECTION 5
EXECUTION ENVIRONMENT

TASK GROUPS AND TASKS

System control of user applications and system functions is accomplished within the framework of the task group, which consists of a set of related tasks. The simplest case of a task can be considered to be the execution of code produced by one compilation or assembly of a source program (after the code is linked and loaded).

The operating system allows the user to configure a system dedicated to online applications or a combination of online and batch applications. This flexibility of configuration is based on the concept of the task group as the owner of the system resources it requires for execution.

By defining more than one application task group to run concurrently, the user can achieve multiprogramming. He can step through an application in sequence, by causing tasks in the group to be executed one at a time; or he can multitask an application, by causing tasks within the group to be executed concurrently.

Since multiple applications can be loaded in memory at the same time, contending for system resources, an environment must be defined for each application so that it knows the limits of its resources. This defined environment is called a task group, whose domain includes one or more tasks, a memory pool, files, peripherals, and priority levels. By defining the total system environment to consist of more than one task group, the resources are divided up so that more than one application can run concurrently. For Example, memory is divided into memory pools, and task code of a task group is loaded only into that task group's pool and obtains dynamic memory from that pool.

By using the resources of one task group repetitively, an application can be run as a sequence of job or program steps. To achieve this, a task group can be created by a SPAWN GROUP command to use the command processor, whose function is to process system-level commands. The command processor is activated as the lead task of the task group. One method of sequencing the steps of an application task group is to submit a command to the command processor to read an application command file containing a sequence of names of bound units (files of executable code), where each bound unit corresponds to a program. When a bound unit name is encountered in the file, that bound unit is loaded and executed before the next bound unit name is read. The following is an example of bound unit names in a command file:

```
REP_DATA (The name of a program that gathers report data)
PR_RPT   (The name of a program that prints the report)
```

Another method of sequencing application steps is to issue a SPAWN TASK command for each task to be executed. The SPAWN TASK command causes the task to be loaded, executed, and then deleted. Provided the command processor is instructed to wait for completion of each spawned task, the tasks in the task group can be executed in sequence. For example:

```
ST 1 -EFN REP_DATA -WAIT (Spawn a task to gather report data)
ST 1 -EFN PR_RPT -WAIT (Spawn a task to print the report)
```

Since the Level 6 can be thought of as a set of processors – the central processor, each input/output device, and the real-time clock – the above procedure can be used to attain another effect, that of multitasking within one task group. Consider the situation when the command processor is the lead task, and it reads a file containing SPAWN TASK commands; it does not wait for the execution of the individual tasks, but continues to spawn tasks until it reads an end-of-file or &Q directive. All these spawned tasks are loaded and run

concurrently in this task group, contending among themselves for the resources defined for the task group. For example:

```
ST 1 -EFN REP_DATA (Spawn a task to gather report data)
ST 1 -EFN PR_RPT (Spawn a task to print the report)
```

The command processor does not have to be the lead task of a task group. An application consisting of one task could execute in a task group whose lead task is the application task. Should your application require step control or multitasking, but you do not need the control through commands, you can generate a task group whose lead task contains assembly language system service macro calls whose functions are analogous to the CREATE and SPAWN commands.

The above situations are illustrative and do not exhaust the various ways that you can control program execution.

To summarize: a task group is the owner of system resources, and the context in which system control of tasking is accomplished. A task may be characterized as the *execution* of a sequence of instructions that has a starting point and an ending point, and performs some identifiable function. It is the unit of execution of the operating system, and its execution must be requested through the Monitor software.

The source language from which task code is derived may be any of the languages supported by the operating system. Source code is compiled (or assembled) and linked to form bound units consisting of a root and zero or more overlays.

Application Design Benefits of Task Group Use

Designing an application around the task group provides:

- o Intertask communication
- o Operating system control of multiple, unrelated task groups

Intertask Communication

The tasks in a task group execute asynchronously in response to the interrupt-driven nature of the operating system and to a linear scan of priority levels assigned to each task group. Tasks communicate through the control structures supplied with each request for task execution.

Asynchronous tasks provide effective software response to information received from real-time external sources such as communications or process control systems. Usually, the task that is activated to handle the interrupt from the external source has a higher priority and a shorter execution time than the task that processes the information. The task that responds to the interrupt will use the operating system to request the execution of the processing task, supplying along with the request the control structure containing a pointer to the new information to be processed. The operating system responds to the request by activating the requested task, or by queuing the request if there are other requests for the execution of this task still pending.

Communications applications would have a high priority task to examine data received at random intervals and decide which processing task should handle the data. This high priority task uses the operating system to queue requests for the processing task, thereby accommodating peak-load conditions in which data is received faster than it can be processed.

In a process control system, the real-time clock might provide the interrupt that causes the high priority task to scan and update temperature, thickness, or raw material level sensors that monitor the physical status of the process. This information would then be passed to a processing task that determines the necessary adjustments based on the new data. Then a third task, having a priority between the other two, could be requested to make whatever changes are required; for example, to change the flow rate of material entering the process by closing a valve.

These two brief examples illustrate the value of priority assignments and communication facilities between tasks.

Operating System Control of Task Groups

Operating system control of an application based on the use of multiple task groups is important for several reasons. First of all, these applications can be thought of as consisting of multiple, unrelated "jobs" (task groups) made up of one or more "job steps" (tasks). The sequence of task execution can then be controlled by the operating system (command processor) as it processes synchronously-supplied commands instead of responding only to externally-supplied interrupts. The next "step" is started only when the previous step terminates.

Furthermore, if any one set of tasks does not fully use the available processing time, the operating system can make more efficient use of system resources by rotating their use on the basis of interrupts and priority level assignments.

Finally, the use of independent task groups that are subject to operating system control prevents one task group from adversely affecting another. If an error occurs in one task group, it can be aborted while others continue to execute.

To summarize, operating system control of multiple task groups provides these advantages:

- o "Job" and "step" execution sequencing
- o Efficient system resource use
- o "Job" independence

Generating Task Groups and Tasks

The operating system provides tasking facilities regardless of the source code in which the application is written. Once generated, all tasks are subject to the same system controls whether written in COBOL, FORTRAN, RPG, or assembly language. Because COBOL and RPG do not provide for "tasking" as part of the language syntax, the generation of tasks consisting of code written in those languages is done via commands. Although tasks written in assembly language or FORTRAN can be generated at the control language level, these languages have a facility for generating task groups and tasks (FORTRAN) without recourse to commands. Assembly language programs use system service macro calls; FORTRAN has tasking routines.

From the overall system viewpoint, the actions of the control language in the generation of task groups and tasks are much more visible than the same capabilities in assembly language, and will be considered next.

As shown in Table 5-1, commands submitted by the operator and commands submitted by other users share some of the task group generation functions and also perform unique functions. The control commands are in three groupings:

- o Commands that perform the same function whether submitted by the operator or another user (an exception being the group creation/deletion commands in the batch mode)
- o Commands entered only by the "system operator"
- o Commands contained within the content of an existing task group request

Characteristics of Task Group and Tasks

Task groups and individual tasks can be originated in either of two ways: they can be "created" or "spawned." The choice depends on application design considerations as well as the intended functions.

There are important differences between tasks (and task groups) that are generated by a create function, and those originated by a spawn function. Created task groups and tasks are permanent; they remain available in memory until they are explicitly removed. Spawned task groups and tasks are transitory; they perform a function and disappear.

Created task groups and tasks are passive; they must be explicitly requested to execute in order to perform their intended function. Spawned task groups and tasks cannot be requested. The spawning of a task group or task is equivalent to a create-request-delete sequence of control language commands: the task group or task is defined, provided with system resources and control structures, executes, terminates, and has its resources deallocated, all in one continuous process.

TABLE 5-1. TASK GROUP AND TASK FUNCTIONS POSSIBLE FROM ONLINE OR BATCH DIMENSIONS

	Commands		Operator Commands	
Command	**Online**	**Batch**	**Online**	**Batch**
Create Group	Yes	No	Yes	Yes
Enter Group Request	Yes	Yes	Yes	Yes
Delete Group	Yes	No	Yes	Yes
Abort Group	Yes	No	Yes	Yes
Spawn Group	Yes	No	Yes	Yes
Bye	Yes	Yes	No	No
Enter Batch Request	Yes	Yes	NA	Yes
Suspend Group			Yes	NA
Activate Group			Yes	NA
Abort Group Request	Only operator		Yes	NA
Suspend Batch Group	commands exist for		NA	Yes
Activate Batch Group	these functions		NA	Yes
Create Batch Group			NA	Yes
Delete Batch Group			NA	Yes
Abort Batch Request			NA	Yes
Abort Batch Group			NA	Yes
Create Task	Yes	Yes	No operator commands	
Delete Task	Yes	Yes	exist for these	
Enter Task Request	Yes	Yes	functions	
Spawn Task	Yes	Yes		

[a]The command processor executes in both online and/or batch dimensions.
[b]NA means not applicable.

FORTRAN or assembly task code may cause extensive action in its own behalf, as when application task code requests a system service or the execution of another task while awaiting the completion of the requested task. Each task that requests another supplies the address of a control structure through which the issuing task and the requested task can communicate, and which the Monitor software uses to coordinate task processing.

Task Group Identification

Each task group has its own unique identifier. Honeywell-supplied task group identifiers begin with a $. The system task group identifier is $S; the batch task group identifier is $B; the Debug task group identifier is $D. The identifier for an online task group, specified in the create or spawn group command, is a 2-character name that should not have the $ as its first character. The identifier (or group-id) may be indicated or implied in commands to designate what task group is to be acted upon. The operator may include the task group identifier in responding to messages from the task group.

MEMORY USAGE

The operating system manages a memory configuration that consists of two different areas of available main memory: an online dimension and a batch dimension. The online dimension is subdivided into the system and one or more online pools, which may belong to more than one task group. The batch dimension is a single memory area or memory pool that belongs to the batch task group.

Memory Layout

The Configuration Load Manager (CLM) reads a directive file, and from the specifications supplied, it sets up memory pools and indicates to the loader what system and user-written software is to be resident for the life of the system.

The numbers and sizes of memory pools are specified in MEMPOOL directives to the CLM. The system and batch pools are defined by an "S" or a "B" on their respective MEMPOOL directives; all other pool definitions are application online pools.

Figure 5-1 shows a memory configuration consisting of resident operating system (and possibly user-written) software beginning at location zero, the system pool adjacent to it, three online pools for use by application task groups, and the batch pool which is always located at the highest memory address. Online and batch pools are described below, followed by a summary of the contents of the operating system area and the system pool area.

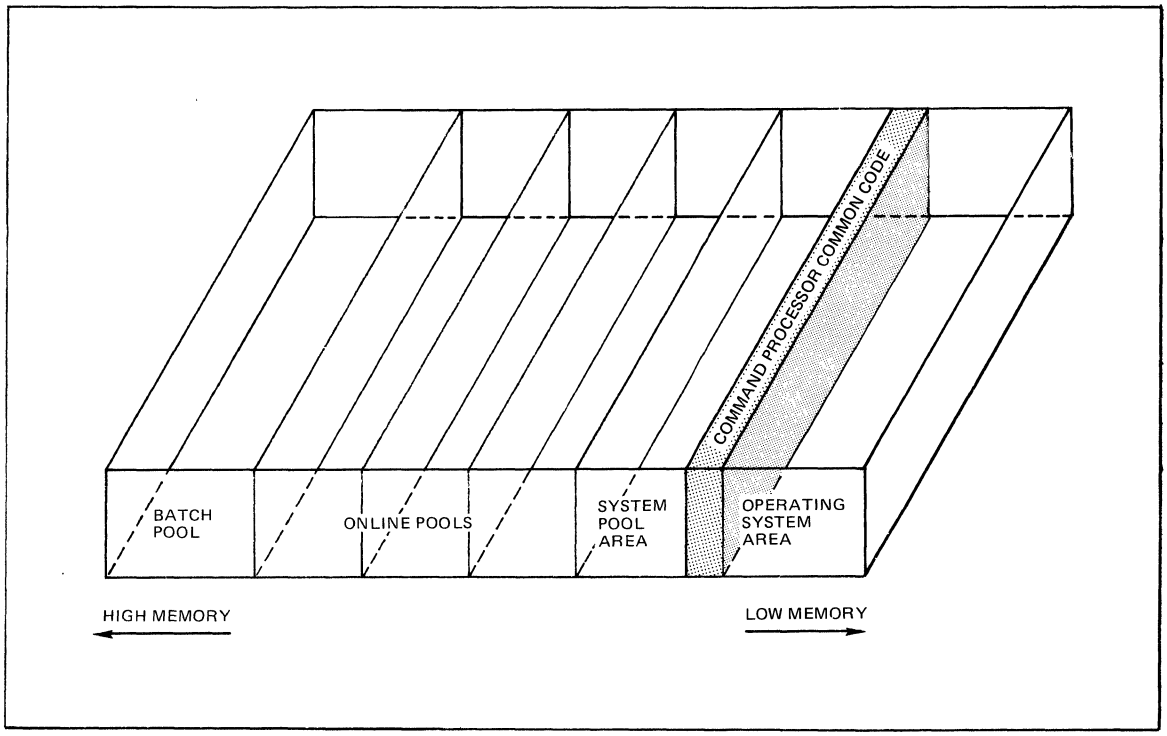


Figure 5-1. Memory After Configuration

Online Pools

Online memory pools are defined in MEMPOOL directives submitted to the CLM during configuration. The definition of online pools requires careful consideration based on the following facts:

- The operating system acquires space on behalf of tasks of a task group from the pool belonging to that task group—not from the system pool (for the exception to this convention see "Sharable Bound Units" later in this section). This means that work space for the task and space for some of the file control and other data structures must be included in the calculation of the task's memory pool size. See the *System Building* manual for these size calculations.
- Online pools can be shared by more than one task group.
- The batch pool can be rolled out to accommodate the extension of an online pool into the batch pool area.

There are two types of online pools, exclusive and nonexclusive pools. Online pools can also be specified as expandable and may expand into the batch pool space, thus forcing a rollout of the batch task group. Expandable online pools need not be contiguous with the batch pool.

Calculation of the desired size of system pools is necessary before a trial configuration can be made for an application. However, these calculations may be rough approximations in the early stages of application development. Both the system pool and batch pool (if any) must be explicitly defined in size. The configuration process allows one physical area of memory in the online pool area to be defined as having a size equal to the remaining memory available (* convention) without making precise calculations. When the * convention is used for an exclusive pool definition, nonexclusive pools may not be defined. The * convention for nonexclusive pools is illustrated below.

Exclusive Online Pools

An exclusive pool is one whose boundaries do not overlap those of other pools. An exclusive pool is defined in a MEMPOOL directive to CLM with "E" as the first parameter. The lower part of Figure 5-2 shows a configuration of five online exclusive pools. Each pool is to be used for the tasks of one task group. The pools are shown "empty" as they would be at the end of the configuration process. The only task code that is loaded into pool areas as a result of system configuration is the command processor residing in the system pool area. All other bound units that execute as tasks are loaded in response to specific commands to the command processor.

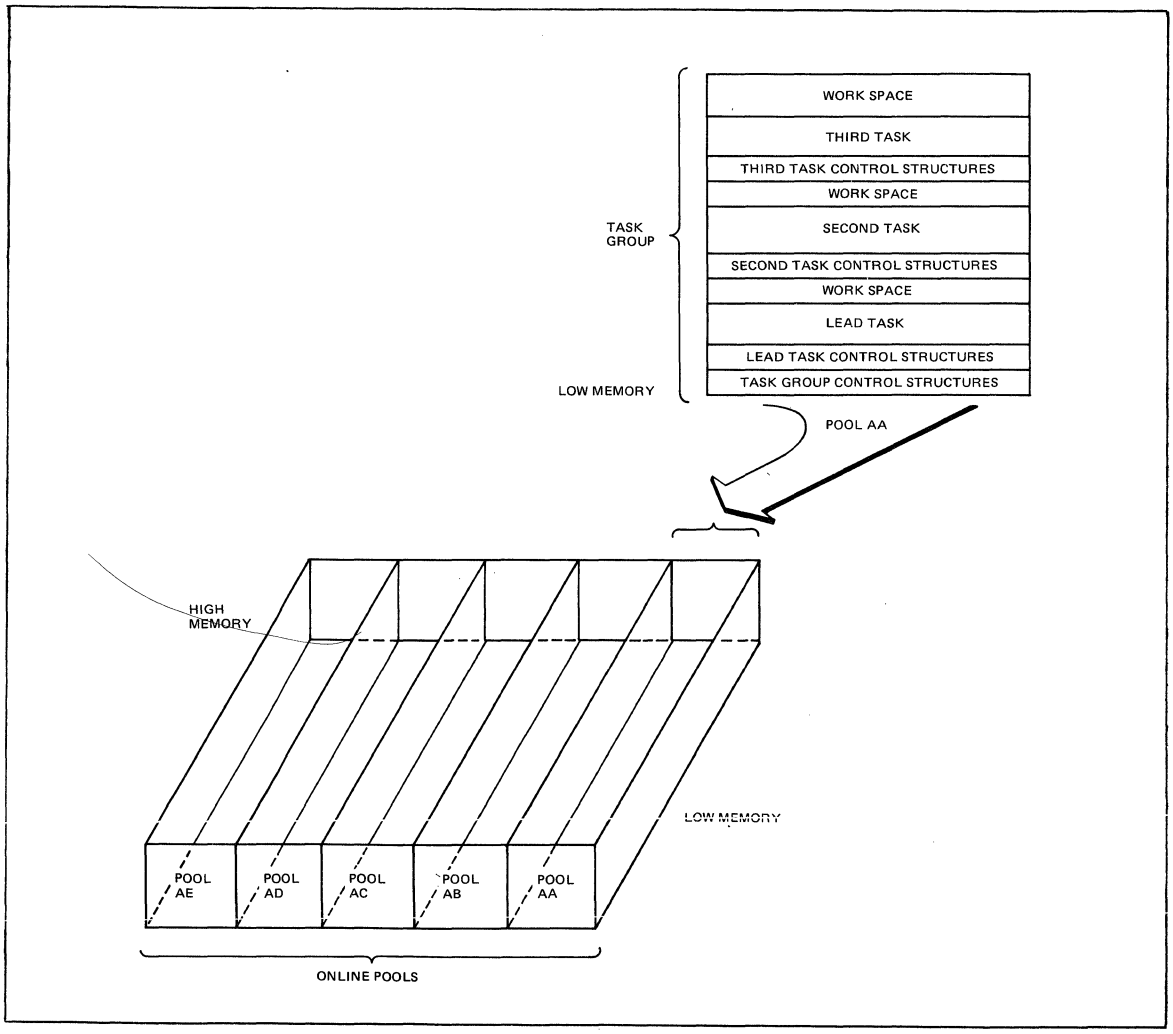


Figure 5-2. Exclusive Memory Pools and Contents

The upper part of the figure is an idealized picture of the contents of pool AA at some instant during processing and shows the memory layout for three concurrent tasks. The figure does not accurately reflect the fact that memory is allocated and returned (in assembly

language programs) dynamically when needed, and work space might not be contiguous to the task code that requests it. Also, memory is allocated in contiguous blocks according to an algorithm that uses multiples of 32 word blocks to calculate the amount of space to assign to a pool. So the "holes" that would normally be present as a result of the operation of the algorithm are missing. The memory manager adds two words for control information to the requested amount of space, divides this value by 32, rounds up to the next whole number, and allocates this calculated amount of memory to a task.

The tasks in pool AA (and in all task groups) have priorities assigned to them that are relative to the priority of the lead task, eliminating conflict for resources.

The generation of task groups to use the pools in the figure could be carried out entirely from the operator's terminal or the system START-UP EC command file using the command processor. Alternatively, after creating a group whose lead task is the command processor, that group could be used to generate task groups and tasks by reading its command file.

The characteristics of exclusive pools:

- o Have "E" as the first parameter on a MEMPOOL directive
- o Are an explicit size (the last pool specified need not be given an explicit size)
- o Consist of one or more sets (all pools are defined by one or more MEMPOOL directives)
- o Are allocated sequentially beginning at the system pool
- o Can be expandable (can cause roll-out of the batch task group)

Nonexclusive Online Pools

A nonexclusive pool set is a set of pools whose boundaries overlap those of other nonexclusive pool sets so that some memory locations are common to both pool sets. Figure 5-3 shows two pool sets which are really alternative definitions of the same physical memory area.

Nonexclusive pools are defined in MEMPOOL directives to CLM with a blank first parameter. The characteristics of nonexclusive pools:

- o Have a blank first parameter on a MEMPOOL directive
- o Are an explicit size (except that the last pool in each pool set need not be given an explicit size)
- o Can consist of more than one set
- o Are allocated between the exclusive pools and the batch pool
- o Can be expandable (can cause roll-out to the batch task group)

Sharing Memory Pools

There are two ways of sharing memory pools. The first method involves assigning two or more task groups to the same pool. As these tasks execute, they contend for the same memory space. Therefore, they should be designed so that they can be suspended or take some alternative action when no additional memory is available.

The second method of allowing task groups to share memory involves the definition of nonexclusive pool sets. Figure 5-3 shows how this might be done.

Pool sets SX/SW/SV and SZ/SY represent alternate definitions of the same physical area of memory. This method of memory use has the advantage of providing flexible and efficient use of resources at any one time. But it has the disadvantage that, unless task group requests are very carefully planned, software deadlock (memory usage conflicts) can occur – the operating system does not prevent it. For example, if task groups using the SZ/SY pool set never execute until those using the SX/SW/SV pool set have terminated, there will never be a problem of deadlock (if only one task group uses each pool).

However, assume that a task group assigned to pool SY is generated and acquires some of the pool space. Then, a task group assigned to pool SW is generated and acquires some space. If each group requested its remaining space, and was willing to wait until the space was available, a deadlock would occur–neither task would ever complete.

The surest way to avoid potential memory usage conflicts is to define all online pools as exclusive pools, and additionally to confine pool use to *one* task group.

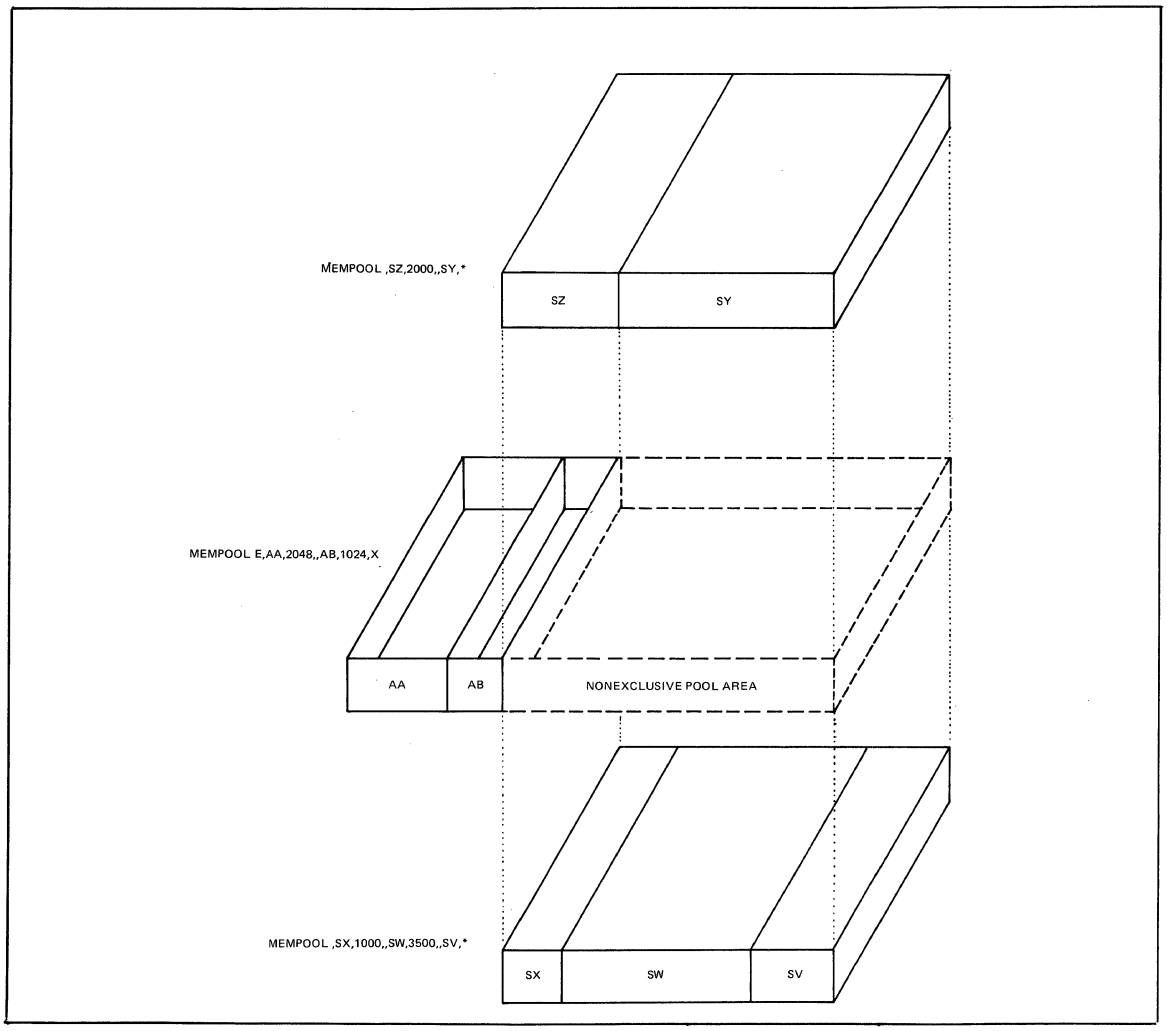


Figure 5-3. Exclusive and Nonexclusive Pool Sets

Batch Pool and Roll-Out

If you configure a system to include a batch pool (by specifying "B" as the first MEMPOOL parameter), it is a resource of the batch task group, whose lead task is the command processor. The usual use for the batch dimension is for program development programs that can be rolled out to provide additional memory for online tasks.

Tasks executing in the batch pool are subject to roll-out when a task executing in an extendable online pool exhausts its assigned memory pool. The operating system initiates roll-out of the batch pool as soon as all batch task group I/O transfers are complete. If no I/O operations are in progress, the task group will be suspended immediately and the entire pool area written out to the roll-out file (>SID>ROLLOUT) on the bootstrap disk as one large record.

Online tasks that use the memory extension capability must be designed so that they can wait for roll-out to occur. Furthermore, unless online task memory requests are coordinated, they could cause "thrashing"—the rapid roll-out/roll-in of the batch pool. Once roll-out is started, it will complete before roll-in occurs, even though the condition that caused roll-out has been removed. For example, Task A requests memory; none is available in its pool; roll-out is requested; Task B immediately releases a block of memory in the same pool large enough to accommodate Task A's request. Roll-out will proceed anyway. Roll-in is initiated when the expanded memory usage is over only if no explicit roll-out request is received.

Operator commands can be used to cause roll-out and roll-in: the SSPB (suspend batch) command causes execution of the batch task group to be terminated temporarily and the group to be rolled out of memory; the ACTB (activate batch) command causes the suspended batch task group to be rolled back into memory and execution to be resumed.

Batch Task Group

The properties of the batch task group and its tasks are as follows:

- o The lead task is always the command processor.
- o Tasks can only refer to peripheral devices, or mass storage directories and files that are marked shareable.
- o Task code cannot execute privileged central processor operation codes (I/O, HALT, LEV).
- o On a model 6/43 with a Memory Management Unit (MMU), tasks cannot alter memory outside the batch pool.
- o Tasks cannot issue system service macro calls or commands that alter the set of task groups defined to the operating system, e.g., CREATE GROUP, DELETE GROUP, ABORT GROUP, SPAWN GROUP.
- o A monitor I/O read request will have its boundaries verified that they fall within the batch pool.
- o Real-time application tasks should not be scheduled for execution under the batch task group if the batch pool is subject to roll-out.

Operating System Area

In response to CLM directives, the following software components and data structures will be located in the fixed operating system area after the configuration process is complete:

- o Basic operating system software plus resident overlays (RESOLA directive)
- o User-written extensions to operating system (LDBU or DRIVER directive)
- o Device-drivers
- o Intermediate request blocks needed for task groups (SYS directive)
- o Trap save areas (SYS directive)
- o Overlay area(s) for system software (SYS directive)
- o File control structures (file description block (FDB) for nondisk devices)

The operating system area is fixed–its contents remain the same for the life of the system – in contrast to other memory areas whose contents can vary. Almost all code loaded into this area is reentrant so that a single copy of the code is available to multiple users, thus minimizing memory requirements.

System Pool Area

The area adjacent to the resident software area is called the system pool. This area contains the system task group. In addition, the system pool accommodates the following elements:

- o Current function invoked by an operator command
- o Extended trap save areas (TSAs) needed during processing
- o Control structures for the batch task group
- o Shareable bound units
- o File system directory and file definition blocks

System Task Group

The system task group differs from other task groups in the following ways:

- o Cannot be aborted or suspended
- o Always has read and write access to all of memory

- o Handles all system dialog (including operator commands) through the designated operator terminal
- o Never terminates, so it cannot be requested

Batch Task Group Control Structures

The following control structures are found in the system pool area whenever a batch memory pool is configured:

- o Group control block (GCB)
- o Logical resource table (LRT)
- o Logical file table (LFT)
- o Task control block (TCB)
- o Batch request block

File Control Structures in the System Pool Area

The elements in the system pool area that are used for file control consist of:

- o File description block (FDB)
- o All buffer control blocks (BCB)
- o Buffers for shareable files

BOUND UNITS

Task code is derived from the source language of programs that are compiled or assembled to form object units. One or more object units are linked to form a bound unit that is placed on a file. The bound unit is an executable program that can be loaded into memory. A task represents the execution of a bound unit. Each bound unit consists of a root segment and any related overlay segments.

Overlays

To minimize the amount of memory required to execute a bound unit containing application code, the bound unit can be created as a root and one or more overlays. Object units whose code is to be loaded as overlays are defined as overlays by the Linker. The use of overlays requires careful planning so that required code is not lost or repetitively loaded.

Nonfloatable and Floatable Overlays

Overlays are part of a bound unit which comprises a root or a root and one or more overlays. There are two types of overlays: the nonfloatable overlay that is loaded into the same memory location relative to the root each time it is requested, and the floatable overlay that is linked at relative location 0 and can be loaded into any available memory location.

Floatable overlays must have the following characteristics:

- o External location definitions in the overlay are not referred to by the root or any other overlay.
- o The overlay makes no immediate memory addressing (IMA) references to itself, and no displacement references to the root or any other overlay.
- o The overlay can contain IMA references with or without offsets to the root or any other nonfloatable overlay.
- o The overlay does not contain external references that are not resolved by the Linker.
- o The overlay must be linked *after* all nonfloatable overlays have been linked.

A user program can use one or more areas of its available memory for placement of floatable overlays. To be most effective, the program must perform its own memory management of these areas. To perform memory management, a user-written assembly language overlay manager must be linked with the root of the program bound unit. Adequate memory space must also be provided in the pool of the requesting task. If the user program does not control the placement of a floatable overlay, the system will place the overlay in available space in available memory.

Assembly language programs can use system service macro calls to load and execute nonfloatable overlays; memory management is handled by the Monitor. Similarly, COBOL programs can use CALL/CANCEL statements to control nonfloatable overlays. FORTRAN and RPG programs must link a user-written assembly language overlay manager with the application program.

Resolving References

Forward references can be made to symbols defined in object units to be linked later. Backward references can be made to symbols previously defined provided that the defined symbols were not purged from the Linker symbol table by a Linker BASE or PURGE directive. Since the specification of the BASE directive removes from the Linker symbol table all previously defined, unprotected symbols that are at locations equal to or greater than the location designated in the BASE directive, you must either define all symbols in a nonoverlaid part of the root or plan the linking of subsequent overlays so that purging of needed symbols does not occur.

Floatable overlays can refer to fixed addresses in the root or nonfloatable overlay, but cannot refer to addresses in another floatable overlay.

When a root or an overlay of a bound unit is loaded, the loader examines the attribute tables associated with the bound unit, if an alternate entry point is specified. The loader tries to resolve any references to symbols that remain unresolved at load time by searching the system symbol table (i.e., the resident bound unit attribute table); it cannot resolve any references to symbols that do not exist in that table (Linker symbol tables do not exist at load time).

Sample Overlay Layout

Figure 5-4 illustrates the layout of overlays in a memory pool AA. The Linker directives to create and specify the location of these overlays are described in the *Program Execution and Checkout* manual.

When the root is loaded, the largest contiguous amount of memory necessary to accommodate the *root* and all *nonfloatable* overlays is allocated. Except for space for any floatable overlays, no other memory requests need be made. In the figure, this memory area begins at relative 0 of the root, and continues to the end of object unit OBJD. The root consists of object units OBJ1 and OBJ2. When loaded, OBJ5 of overlay ABLE will replace the previously loaded OBJ2 code of the root. Similarly, the overlay locations were specified so that OBJC of overlay ZEBRA will replace part of OBJB.

Shareable Bound Units

Using shareable bound units is a way of minimizing application task group memory requirements while making reentrant code available to multiple tasks. Unlike permanently resident bound units that are loaded during system configuration, shareable bound units are transient in the system pool and are loaded during processing. A counter is incremented each time a request is made for the bound unit, and the unit remains in memory as long as a task is using the code. As soon as the counter is decremented to zero, the system pool space occupied by the bound unit is returned to available status.

Operator commands can be used to load and then unload a shareable bound unit.

To be recognized as shareable by the loader and loaded into the system pool, the bound unit must have been so marked by the Linker in response to a SHARE directive when the bound unit was linked. If the system pool does not have enough space to accommodate the bound unit, at a given instant, it will be placed in the pool associated with the task group that requested the bound unit, and cannot be shared.

Shareable bound units and the operating system extensions that are loaded when the system is configured differ in another way. Namely, operating system extensions can be referred to directly by any task, but a shareable bound unit must be accessed as a task. The reason for the difference is that an operating system extension is loaded when the system is configured—its symbols are included in the system symbol table at that time. Since there is no symbol definition once configuration is complete, and since a shareable bound unit is loaded *after* the system has been configured—no entry for it exists in the system symbol table, and it must be accessed as a task.

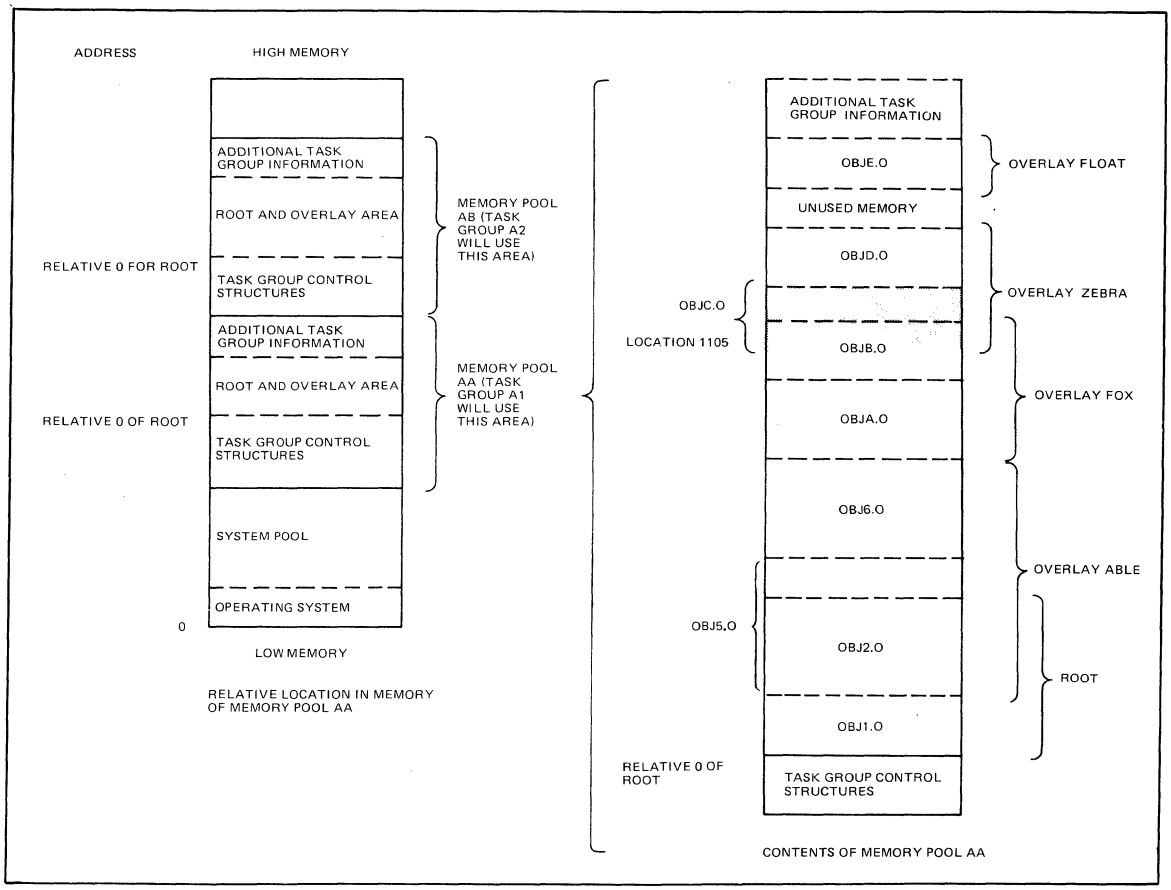


Figure 5-4. Overlays in Memory

Table 5-2 compares permanently resident operating system extensions and transient shareable bound units.

TABLE 5-2. COMPARISON OF OPERATING SYSTEM EXTENSIONS AND SHAREABLE BOUND UNITS

Characteristics	Operating System Extension	Shareable Bound Units
Multiple Users	Yes	Yes
Permanent Resident (fixed area)	Yes	No
Temporary Resident (dynamic area)	No	Yes
Symbols in System Table	Yes	No
Accessed symbolically?	Yes	No
Accessed as a task?	Yes[a]	Yes
Can have overlays?	No	Yes
Called by Bound Unit Name	No[b]	Yes

[a]If the extension is an assembly language bound unit, it may have within it sections of code or control structures controlled by semaphores which would be accessible to other assembly language tasks.

[b]The operating system does not "remember" extensions by their names; a request for one by name results in another copy being brought into memory.

Loading Bound Units (Search Rules)

The loader follows preestablished search rules when searching through a set of directories to locate a bound unit to be loaded. The loader initiates the search in response to a command which contains an argument naming the bound unit to be loaded.

Search rules that regulate the search process define three directory pathnames and the sequence in which they are used during a search. They are:

- o The task group's working directory
- o System directory -LIB1 argument of the CSD (change system directory) command
- o System directory -LIB2 argument of the CSD command

At the completion of command processor start-up, the pathname of the system task group's working directory is ^system_volume_name, and remains so until modified by one or more CWD (change working directory) commands. Both system directories are initially set to >SYSLIB1. The system STARTUP.EC file executed immediately after configuration should initially be used to change the value of the second system library searched. It would normally be changed to >SYSLIB2 on a cartridge disk or storage module system and ^ZSYS01>SYSLIB1 on a diskette system. The operator command CSD (change system directory) may be used to change pathnames associated with system directory arguments -LIB1 and -LIB2. The pathname of a user task group's working directory is established through a CWD command or through the -WD argument in the EBR (enter batch request), EGR (enter group request), or SG (spawn group) commands.

SECTION 6

TASK EXECUTION

A task may be characterized as the execution of a sequence of instructions that has a starting point and an ending point and performs some identifiable function. In an assembly language program, a task can initiate another task for execution or terminate itself by calling task management functions. Multiple tasks can operate independently of and asynchronously to each other.

Each application, system, or device driver task operates at an interrupt priority level, one of the 64 priority levels provided by the hardware/firmware. This section describes the processing of priority levels, including context saving of interrupted tasks, and the assignment of priority levels and logical resource numbers to tasks. Communication between tasks, task coordination, and task handling by the system are also summarized in this section.

INTERRUPT PRIORITY LEVELS

All system tasks, device driver, and application tasks are assigned interrupt priority levels that indicate the order of their execution. Control of the central processor is given to the highest active interrupt level. An overview of the priority levels, a description of how the hardware/firmware processes priority levels, and information on controlling levels (the latter of interest primarily to the assembly language programmer) are given below.

Processing Priority Levels

A Level 6 central processor provides 64 potential interrupt priority levels that are used by the hardware to order the processing of events. These levels are numbered from the highest priority (level 0) to the lowest priority (level 63). Levels 0 through 4 are reserved; level 63 is the "system idle" level; the intervening levels (5 through 62) are assigned to logical resources, i.e., devices and tasks.

The determination of which priority level is to receive central processor time is based on a linear scan of the level activity indicators. The level activity indicators are maintained by the hardware in four contiguous dedicated memory locations (see Figure 6-1). Each bit that is "on" denotes an *active* priority level; each bit that is "off" denotes an *inactive* level.

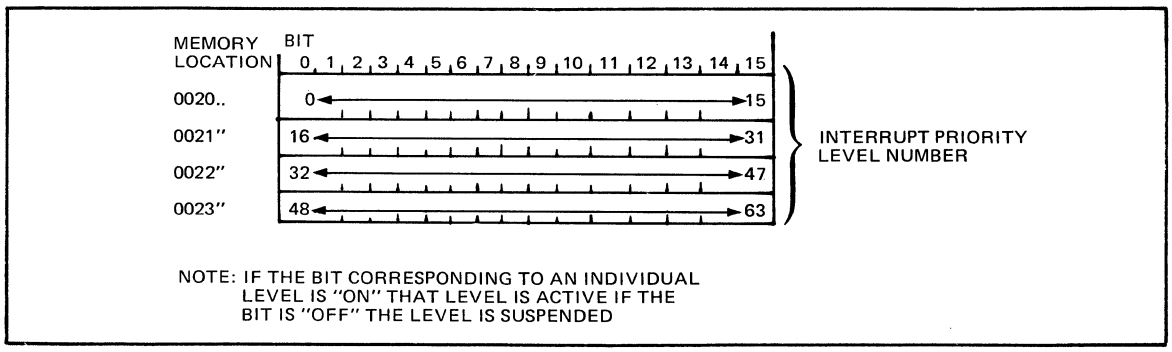


Figure 6-1. Format of Level Activity Indicators

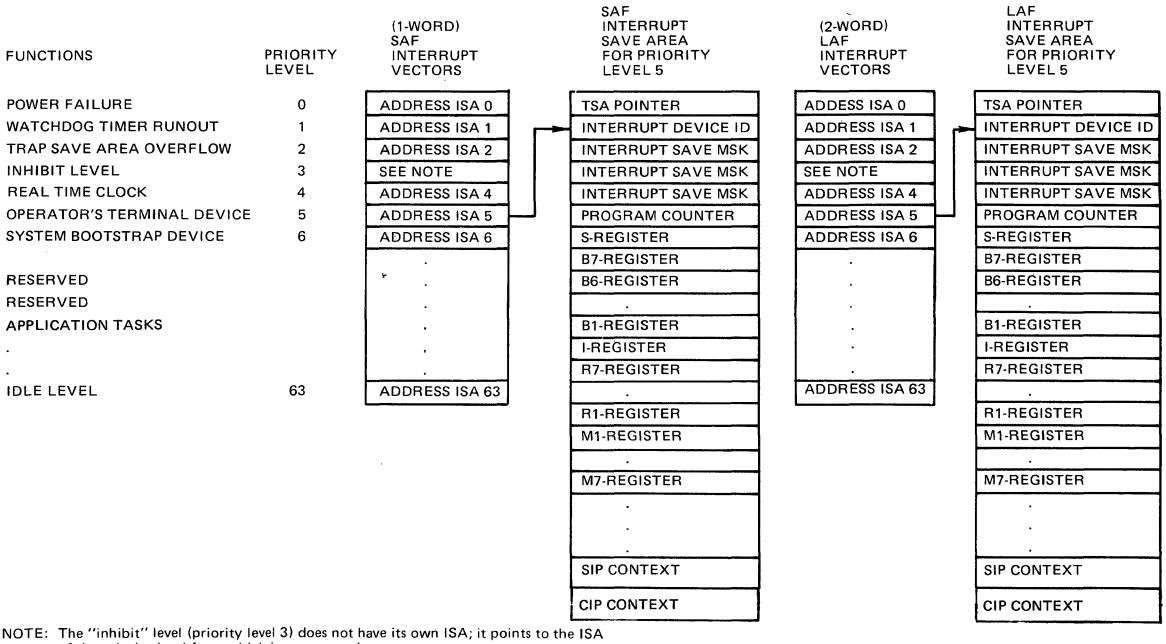
When a given priority level is the highest active level, it receives all available central processor time until it is interrupted by a higher priority level or until it relinquishes control by suspending itself (setting its level activity indicator "off"). If a priority level is interrupted by a higher priority level, its level activity indicator remains "on" and it will resume execution of the interrupted logical resource when it again becomes the highest active priority level. Each time a priority level change occurs, the hardware/firmware saves the context of the *previously* highest active level and restores the context of the *new* highest active level. Inter-

rupting a task, saving the context of a task, selecting and starting the highest priority level task, and restoring the context of a task are done without software involvement.

When more than one logical resource is assigned the same priority level, software determines in round robin fashion, the next active logical resource to resume execution at this level. Thus a task does not block a level when the task is put in a wait state after a request to wait, wait on list, request semaphore, terminate, or after a Monitor call that does a wait for a data transfer. Another task on the same level that is ready will be activated.

Interrupt Save Area (ISA)

The context of a level can include the contents of the program counter, the S-register, all B-registers, the I-register, all R-registers, all M-registers and all SIP and CIP registers. The context is stored in a block of memory known as an interrupt save area. The hardware/firmware context save/restore function finds the appropriate interrupt save area through a pointer supplied in the interrupt vector for that level. The interrupt vectors are a set of contiguous memory locations containing an entry for each potentially active priority level and ordered by ascending priority level number. Figure 6-2 illustrates the order of the priority levels, their corresponding interrupt vectors, and the format of an interrupt save area.



NOTE: The "inhibit" level (priority level 3) does not have its own ISA; it points to the ISA of the priority level from which it was entered.

Figure 6-2. Order of Interrupt Vectors and Format of Interrupt Save Areas (SAF/LAF)

The three highest priority levels have dedicated assignments of special hardware/firmware functions (viz., incipient power failure, watchdog timer runout, and trap save area overflow). Priority level 3 is reserved as an inhibit level and level 4 is dedicated to the real-time clock. Succeeding levels are configured as device levels. Following these are two levels that are reserved for system use. Except for level 63, the remaining levels can be used for application tasks. Level 63 is reserved for an always-active software idle loop.

Control of Priority Levels

The operating system controls multiple levels through the use of the LEV (Level Change) instruction, which provides the following functions:

- *Resume* - Ascertain the highest active priority level by examining the level activity indicators. Restore the context of this priority level and continue on this level.
- *Suspend* - Mark the current priority level as inactive (reset the level's activity indicator); save the context of the current priority level; go into resume state. Except for this des-

cription of a firmware state, in this documentation set, the term "suspend" indicates the logical state of a task as described in the paragraph "How the Operating System Handles Tasks."

- o *Activate a Level* - Mark the target priority level as active (set the level's activity indicator); save the context of the current priority level; go into resume state.
- o *Inhibit* - Mark dedicated, high-priority level as active and immediately assume this level. Continue execution on this priority level with no context save or restore.
- o *Enable* - Return to the highest active normal priority level from the inhibit level. If the highest priority level is the same one from which the inhibit level was entered, do not perform a context save or restore.

The number of the highest active priority level can be ascertained by interrogation of the contents of the S-register.

The availability of an inhibit level allows critical sections of code to be executed without danger of interruption. For example, it may be necessary to temporarily assume the inhibit level in order to protect short sequences of instructions that modify data structures shared between levels.

A standard feature of Level 6 central processors is a real-time clock, which interrupts at a preassigned priority level (level 4) each time its specified scan cycle has elapsed.

TRAP HANDLING

The operating system provides a means by which certain events which occur during the execution of a task can be "trapped", or handled by routines which are designed specifically to cover the condition causing the trap. Events such as the detection of a program error, hardware error, arithmetic overflow, or uninstalled optional instruction cause traps, control transfers to designated software routines, to occur.

Traps fall into two classes: standard system traps, for which routines are supplied with the operating system, and user-specific traps, for which the user can supply his own handler.

An application program can designate which traps are to be handled through specification of the enable/disable user trap macro calls (see the *System Service Macro Calls* manual for details). If an enabled trap occurs in the user program, the trap manager transfers control to the connected trap handler for the condition causing the trap. A trap that is enabled is local to a task, that is, it neither affects nor is affected by the handling of the same trap in another task, even within the same task group.

Any trap which occurs when its handler is not enabled, or which does not have a handler to process it, causes the executing task to be aborted.

OPERATING SYSTEM FEATURES AFFECTING TASK EXECUTION

The operating system does not monitor resource use either within a task group or among task groups using the online pools. Tasks and task groups must cooperate in their use of system resources to ensure smooth operation of the application.

Peripheral Device Assignments

DEVICE or DRIVER directives processed by the CLM during configuration identify peripheral devices to the operating system. (Communication devices require an additional device-type-specific directive; see the *System Building* manual for details.) Any online task can use any peripheral device or disk file. A batch task can use any peripheral device or disk file that is marked as shareable. The command MF (MODIFY FILE) can be used to change a peripheral device, a disk directory, or file to nonshareable and back again. MF won't be recognized when issued from the batch task group. If just one directory in the full pathname of a file is marked as nonshareable, that file cannot be used by the batch task group. The MF command is also used to control accessibility (read-only and write-only) to files created by any program, but particularly on behalf of a high-level language program that cannot set concurrency restrictions as a part of the language syntax. Read/write/share permissions can be set directly by an assembly language program.

Priority Assignments for Tasks

Priority levels (5 through 62) are assigned according to application design objectives to system, device driver, and application tasks. Assignment of priorities to system tasks and driver tasks (performed during configuration) and to application tasks (performed during task group creation) are described below.

Assigning Priorities to System Tasks

The priority levels for system tasks cannot be explicitly specified; priority levels for devices are included in the specifications submitted on the DEVICE or DRIVE directives to the CLM when the system is configured. The two priority levels following the last one assigned to a configured device are used by the operating system and cannot be assigned to tasks. Table 6-1 summarizes a priority level assignment scheme for tasks and devices.

The table indicates I/O devices, and not device drivers, to stress that each peripheral device must have at least one level assigned to it; peripherals (other than communications devices) cannot share a level. If there are two printers, each must be assigned a unique level even though there is only one copy of a reentrant I/O driver. Communications requires one non-shareable level (COMM CLM directive) dedicated to processing communications interrupts, and it must be at a higher level than any communications device. Communications devices can share a level. For example, four TTYs and one VIP can either share one level or be configured to use up to five levels. The listed priority arrangement provides maximum throughput for each device by assigning the high-transfer-rate devices a higher priority than the low-transfer-rate devices. The criteria used to specify Table 6-1 might not suit a particular application and the level assignments should be modified to include other priority considerations.

TABLE 6-1. PRIORITY LEVEL ASSIGNMENTS FOR TASKS AND DEVICES

Level	Use
0	Power failure handler
1	Watchdog timer runout
2	Trap save area overflow
3	Inhibit interrupts
4	System clock
	Operator's Terminal Device
	System bootstrap device
	Debug program
	Communications interrupt
	Communications devices (high transfer rate, e.g., 9600 bps)
	Cartridge disks
	Communications Devices (low transfer rate, e.g., 1200 bps)
	Diskettes
	Magnetic tapes
	Printers
	Card readers
	Card punches
	KSR
	Operator interface manager interrupt[a]
	Operating system task[a]
	Input/output - bound application tasks
	Central processor - bound application tasks
63	System idle loop (always active)

[a]Mandatory, reserved for system use.

Assigning Priorities to Application Tasks

Priorities are assigned to user task groups and tasks when they are created or spawned. The command to generate a task group contains an argument that represents the base priority level for the task group. The base priority level is relative to the highest system physical priority level, i.e., the highest interrupt priority level number used by the system in that configuration. When a task group is assigned a base priority level of zero, the lead task group executes at the physical interrupt priority level that is the next level above that of the highest level system task. When other tasks in the same task group are created or spawned, they are given level numbers relative to the base priority level assigned to the task group. The physical interrupt priority level at which a task executes is the sum of the highest system physical priority level plus one, the base priority level of the task group, and the relative priority level of a task within that group. This total must not exceed 62.

User tasks that are to execute in the online dimension are usually given higher priorities (lower level numbers) than those in the batch dimension. Tasks that are I/O bound should be run at a higher priority than tasks that are central processor bound. This permits 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 have a higher priority, the I/O devices would be idle while I/O bound tasks waited to receive central processor time.

Logical Resource Number (LRN)

An LRN is an internal identifier used to refer to task code and devices independently of their physical priority levels. Use of LRNs makes assembly language application task code independent of priority levels so that if circumstances require a change in priority levels, the task code does not have to be reassembled.

Device LRNs

LRNs are assigned to devices in the CLM DEVICE directives when the system is configured. The operator's terminal is assigned to LRN 0 at configuration. The bootstrap device is given an LRN value of 1, and, if an MLCP-connected operator's terminal is configured, the system assigns it an LRN of 2. Figure 6-3 is an example of priority level assignments for devices and system tasks and the related device LRNs.

LRN	LEVEL	
	0	
	3	INT
	4	CLOCK
0	5	OPERATOR'S TERMINAL
1	6	DISK
3	7	LINE PRINTER
4	8	SERIAL PRINTER
5	9	CARD READER
	10	OPERATOR INTERFACE MANAGER INTERRUPT
	11	SYSTEM TASK

Figure 6-3. Example of LRN and Priority Level Assignments to System Tasks and Devices

Application Task LRNs

LRN assignments to application program tasks are not dependent on the system configuration on which the application task group is running. LRNs are assigned to task code within an assembly language application program through specification of the create group/task macro calls, as well as the macro calls that build data structures ($IORB, $TRB, etc.). LRNs can be assigned at the control language level through the use of the commands (including operator commands) for creation of tasks groups and tasks. An LRN for an application task can have any value from 0 through 252. Within a task group, the LRN for each task must be unique. More than one LRN can be associated with the same level. For example, two tasks at level 23 can be assigned LRNs of 28 and 29, respectively.

Logical File Numer (LFN)

Logical file numbers are internal file identifiers that are associated with file pathnames either at the assembly language level, or for high-level languages at the command level, through GET or ASSOCIATE commands.

Inter/Intra Task Group Communication

Whether or not information can be passed between task groups and tasks depends upon the following considerations:

- o Language in which task code is written
- o Use of the same file by more than one task group

Language Considerations

Task code written in assembly language can pass information to other assembly language tasks in the same task group by using variable-length request blocks. (See the *System Service Macro Calls* manual for details about building these data structures.) High-level languages cannot use this mechanism directly, but would require called subroutines written in assembly language.

Use of Common Files

Tasks within the same (or different) task group can communicate via disk files. The only requirement is that the concurrency status be the same for all tasks using the files.

Task and Resource Coordination

Tasks can be coordinated in either of two ways:

- o Through the use of tasking requests
- o Through the use of semaphores

Task Requests

One task can request another to execute asynchronously with it, or the requesting task can later wait for the completion of the requested task. Both tasks have access to the request block provided by the requesting task, and thus can pass arguments between them.

Semaphores

Semaphores support an application-designed agreement among tasks to coordinate the use of a resource such as task code or a file. A semaphore is defined by a task within a task group, and is available only to the tasks within that group.

For each resource to be controlled, a semaphore is defined, and given a two-character ASCII semaphore name. This name is a system symbol recognized by the Monitor, and *not* a program symbol that needs Linker resolution. The agreement is: each requestor of a resource whose use must be coordinated issues appropriate Monitor calls to the named semaphore to request or to release the resource. The task that defines the semaphore assigns the semaphore's initial value. The Monitor maintains its current value to coordinate requestors of the resource being controlled. A requestor obtains use of a resource if the semaphore value is greater than zero at the time of the request. A requestor is either suspended waiting for the resource or notified that no resource is available if the value is zero or negative.

Monitor service macro calls are used to:

- o Define a semaphore and give an initial value ($DFSM)
- o Reserve a semaphore-controlled resource ($RSVSM); this macro call subtracts a resource, or queues waiter for the resource; i.e., it *decrements* the current-value counter.
- o Release a semaphore-controlled resource ($RLSM); this macro call adds a resource, or activates the first waiter on the semaphore queue; i.e., it *increments* the current-value counter.
- o Request the reservation of a semaphore-controlled resource ($RQSM); this macro call queues a request block (SRB) if the resource is not available. This macro call *decrements* the current-value counter.

A semaphore is a gating mechanism, and the initial value given to it depends upon the type of control you want to exercise.

For example, assume that you want to restrict access to a particular resource to a one-user-at-a-time order. The mechanism would work in the following way:

1. Task A defines a semaphore by issuing the macro call:

   $DFSM ZZ

   Omission of the value parameter causes the initial value to be set at 1.

2. Task B now issues a $RSVSM call; the counter is decremented to 0, Task B gets the resource for itself knowing that no other task using the semaphore mechanism is using or can obtain the resource.
3. Task C issures a $RSVSM call; the counter is decremented to -1, Task C is suspended and put on semaphore queue in first-in/first-out (FIFO) order (Task B is still using the resource).
4. Task B issues a $RLSM when it finishes with the resource; the counter is incremented to 0, Task C now gets the resource. After the $RLSM for Task C, the value is 1 again.

Use of resources by more than one user at a time can be arranged by adjusting the initial value of the semaphore, e,g., an initial value of 2 allows two users, a value of 4 allows four users, and so on, depending on the nature of the resource and its intended use.

If it is undesirable for a task to be suspended while a resource is in use, the $RQSM macro call can be used instead of $RSVSM to reserve a resource. $RQSM is an asynchronous reservation request ($RSVSM is a synchronous request) which causes a request block to be queued for the resource, so that the issuing task can do other processing before the needed resource is available.

HOW THE OPERATING SYSTEM HANDLES TASKS

More than one task can be concurrently active under the operating system. For example, in a multiprogramming environment, a task in each of several task groups can be active and compete for system resources. Another possibility can be a multitasking application where several tasks executing under one task group can be active to compete for system resources among themselves and with tasks from other task groups. A COBOL or RPG program executes as a single task. A FORTRAN or assembly language program can include requests to activate more than one task and synchronize their execution; these requested tasks can execute concurrently.

In order for the operating system to sequence the execution of tasks, each task must be assigned to a priority level. Task competition for the central processor resource is governed by the hardware/firmware linear priority scan of level activity indicators. Tasks on the same priority level execute serially in the order in which they are requested. The highest priority active task receives all available central processor time until it is interrupted by a higher priority task or until it waits, terminates, or is placed in hold. As a result of this linear ordering of task priority, care must be taken that high-priority tasks do not consume an excessive amount of central processor time at the expense of lower priority tasks. A task that has a built-in program loop, to wait for an event occurrence, prevents other tasks at the same or lower priority levels from executing.

In an assembly, COBOL, or FORTRAN language program, program loops might not be necessary since a wait function can be invoked by a task, at some point after a related request has been made, to suspend itself and it will be later reactivated at the time of event completion.

It should be noted that all device drivers are considered to be tasks in the above sense; using the File System, buffered device drivers can execute concurrently with tasks. Drivers execute on the priority levels assigned to individual devices and thus have their own contexts. The device drivers provided in the operating system are written in reentrant code and are therefore capable of servicing multiple devices.

A user task becomes active when a command or operator command is issued for it using a SPAWN GROUP/TASK or an ENTER GROUP/TASK REQUEST. FORTRAN programs can also call START or TRNON; an assembly language program can call $RQGRP, $RQTSK, $SPGRP and $SPTSK to activate a user task.

Tasks may exist in any of the following logical states:

- *Dormant* – There is no current request for the task. A task enters the dormant state if it is created but never requested, or a terminate is issued against it. A task remains dormant until a request is placed against it or it is deleted. If deleted, it is erased, memory is deleted, and it cannot be reactivated.
- *Active* – Executing or ready to execute when its priority level becomes the highest active level in the central processor. A task remains active until it waits, terminates, or is suspended. In the active state, task execution might stall by not having task code executed; e.g., the task issues a synchronous input/output order.
- *Wait* – A task is not executing because it may have caused its own execution to be interrupted until the completion of an event such as the completion of a requested task, or until a timer request is satisfied, or until a task releases a semaphore. A waiting task loses its position in the priority level round robin. The following events always result in a wait: an I/O order to disk, magnetic tape, operator's terminal or unbuffered card reader. Task code written in FORTRAN or assembly language will also wait in the following circumstances: a write order to an interactive terminal, or to a printer when a previous write has not completed; a read order issued before the transfer of the current message from an interactive terminal is complete (i.e., CARRIAGE RETURN not pressed). In COBOL, the latter two circumstances result in a wait, if the program is executing its input/output statements in synchronous mode; otherwise, if in asynchronous mode, the result is a FILE STATUS return code value 9I with no waiting.
- *Suspend* – A task is removed from execution by an *external human* action; e.g., the operator enters a SSPG (suspend group) command or a user interrupts a program with a break. The task is activated through another human action; e.g., the operator enters an ACTG (activate group) command or a user enters a command after a break.

To terminate, tasks of assembly language programs must contain a request to terminate ($TRMRQ) call. Compilers provide this call in the object text and it is executed after the user completes execution.

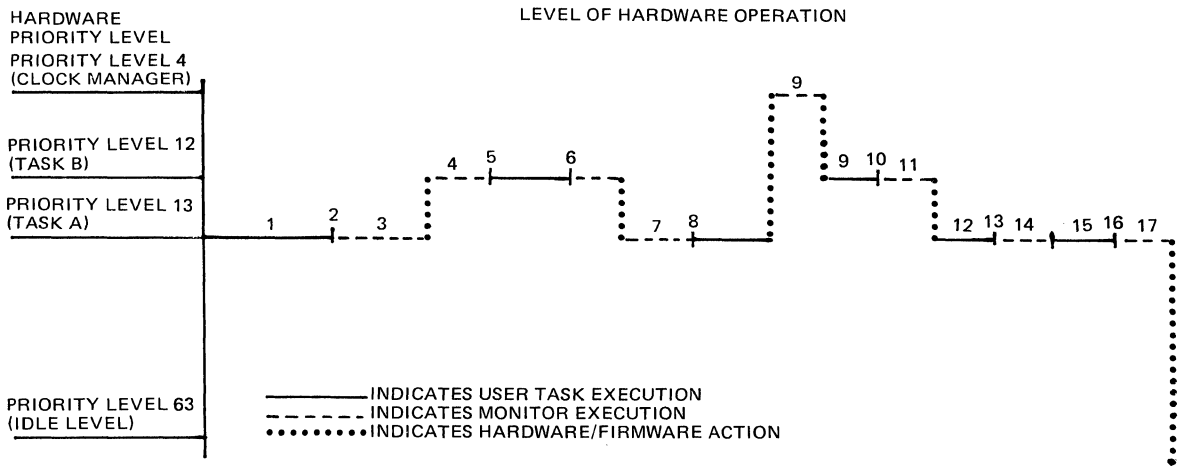
When the concurrent execution of more than one task is desired, each task is specified in a CREATE TASK or SPAWN TASK command or system service macro call.

The procedural code for a requested task is either in a unique bound unit or shared with a bound unit of a task that was previously created. When a task is requested, the Monitor searches the table of LRNs of the current task group under which the task is executing for the identifying LRN, and activates the task, if it is not already active.

EXAMPLE OF MONITOR INTERACTION WITH USER TASKS

The following sequence of events illustrates a typical interaction between the Monitor and two user tasks within a group. For the purpose of this example, task A has an absolute priority level 13 and task B an absolute priority level 12. The absolute priority level is obtained by adding a task's relative priority level, the task group's base priority level, and the highest system physical priority level plus one.

The accompanying diagram indicates the priority levels at which the central processor runs as the sequence of events occurs. The diagram also indicates the consecutive activity of user tasks, the Monitor and hardware/firmware. The numbers in the diagram correspond to the numbers in the sequence of events and are explained in order in the text.



User Task Execution	Monitor Execution
1. Task A is running; task B is requested by task A, or entered or spawned through a command.	
2. Task A does not issue a wait.	
	3. The Monitor places the request in the request queue for task B. (Assume that there are no other requests in this request queue.) The Monitor activates priority level 12.
	4. The Monitor examines the request and ascertains task B's starting address from start address data accompanying the request.
5. Task B begins execution because its priority level is higher than that of task A.	
6. Task B issues a call to the clock manager and issues a wait function to wait for clock time-out. Task B is suspended.	
	7. Now operating at the priority level of task A, the highest active priority level, the Monitor returns control to task A.
8. Task A resumes execution.	
9. Task B's clock-related wait times out. The clock manager interrupts task A. Task B's priority level is activated. Task B resumes execution and continues to completion.	
10. Task B issues a terminate call to the Monitor.	
	11. The Monitor removes the request from the request queue for task B. The Monitor suspends priority level 12.
12. Task A resumes execution because its priority level is now the highest active level.	
13. In a multitasking program, task A could issue a wait call to the Monitor to wait for completion of task B.	
	14. The Monitor detects that task B's request is marked as terminated. Control is immediately returned to task A.
15. Task A continues to completion.	
16. Task A issues a terminate call to the Monitor.	
	17. The Monitor removes the first request from the request queue for task A. If there are no additional requests in this request queue, the Monitor suspends priority level 13. If there are no remaining active priority levels, the Monitor now idles at priority level 63.

SECTION 7
DISTRIBUTED SYSTEMS FACILITIES

The GCOS system is designed to interface with communications and networking products to provide components for implementation of a distributed systems environment. This section describes the following GCOS facilities that use communications software to perform data transfers: the Remote Batch Facility, the Data Entry Facility, and utility programs that support file transmission between the Level 6 and other processors.

REMOTE BATCH FACILITY (RBF)

The Level 6 Remote Batch Facility (RBF) is a software package enabling Level 6 hardware to be used in a remote batch processing environment with Level 66 and Series 6000 host processing systems. Remotely located Level 6 peripheral devices can enter jobs into and receive output from one to four host processors.

The Remote Batch Facility works in conjunction with a host processor and a Front-End Network Processor (FNP), operating under control of General Remote Terminal Supervisor (GRTS) or Network Processing Supervisor (NPS) software.

The Remote Batch Facility can use either of two line protocol conventions that control the flow of data between the Level 6 and the FNP:

- o Remote Computer Interface (RCI)
- o High-Level Data Link Control (HDLC)

The Remote Batch Facility operates under control of the GCOS 6 operating system. Remote batch and GCOS 6 local processing functions that are independent of the host processor can be performed concurrently, provided adequate resources (i .e., memory, peripheral devices) are available. Local processing of the following types can be performed:

- o Program/system development and maintenance
- o User-written processing applications
- o User-written data communications applications

Remote batch terminal (RBT) software is run as a task executing in a unique task group and using the resources reserved for that task group. In systems with adequate resources, the Remote Batch Facility can support the concurrent operation of up to four remote batch terminals. Each RBT permits the batch entry of remote jobs destined for processing in a host system and the receipt of output from those jobs. Each RBT is associated with a stream; i.e., a logical connection that lets data travel between two end points between a device or file at an RBT and a host processor).

In remote batch processing, the following functions can be performed at an RBT:

- o Enter a job or group of jobs for processing by a host processor.
- o Combine input from more than one input medium or file into one job.
- o Obtain the status of jobs in the host processor.
- o Use the transparent binary feature to process cards in nonstandard binary format without performing checksumming.
- o Spool data to a file for temporary storage. Input jobs can be stored on a spool file and later be transferred to the host. Batch output can be spooled and then printed or punched at a later time.

- o Direct that job output be sent to any of the terminal's peripheral devices capable of receiving output, to another RBT, or to the host computer site.
- o Backspace an output file and resume output processing from that point.
- o Change printer forms as specified on GCOS control cards.
- o Abort a file that is being retrieved (output from host) or a job that is being entered (input to host).
- o Restart job output processing, specifying the page and/or line number for printer output or the card number for punch output.

RBF Configuration

The user must configure the system using configuration directives (see the *System Building* manual). He then creates a task group for each RBT, defining initial input and output file assignments, modifying external switches associated with the task group and, finally, invoking the RBT and identifying its processing stream.

Remote Batch Operations

Remote batch operations are controlled by entering commands from either the Level 6 operator terminal or an RBT console. Operator messages are issued on the console and define conditions that may require operator action. If necessary, operator responses can be entered through the console.

The Remote Batch Facility supports multiple communications lines to one to four host processors. The lines can be either all dedicated (which are always connected), all switched (which are connected by dialing a telephone and disconnected at the end of each session), or a mixture of dedicated and switched.

Jobs to be processed can be prepared on cards, and read directly through the card reader by an RBT. Alternatively, the job deck can be prepared using the Editor and read from a file in ASCII code. Or, the cards can be spooled to a file and read in GBCD code.

Output records are delivered to a specified output file or device. If the requested device is in use or inoperable, you can wait for device availability or direct the output to a different device.

Refer to the *Remote Batch Facility User's Guide* for a complete description of the Remote Batch Facility.

DATA ENTRY FACILITY (DEF)

The GCOS 6 Data Entry Facility (DEF) is a multifunctional data entry system. Data can be entered through an operator control station, validated, edited, verified, and communicated to a host computer for further processing.

DEF is a disk system combining Level 6 hardware with DEF software. DEF operates under a task group under GCOS software and can operate simultaneously with other system functions and tasks. It supports up to 12 operator display stations (VIP 7200 terminals) and up to six line or serial printers.

The following functions can be performed using DEF:

- o Develop forms; create, modify, delete, print, and view forms
- o Develop tables; create, modify, delete, print, and view tables. The tables are used to ensure that data is entered correctly or to replace data with other specified data.
- o Enter or modify data records by using a form as a template. The entire form is displayed on the screen, and variable (unprotected) areas can be written into or altered.
- o Verify contents of all or specified fields by rekeying the data.
- o Print entire files or selected records from files. The printout can be formatted or unformatted (all data in a single record is run together on one or more lines).

DEF functions may be run concurrently with other GCOS 6 facilities such as file transmission to and from a host computer.

Interface with Programs

Data entry subroutines can be created to include capabilities such as arithmetic functions (e.g., batch totals) or additional data validation and editing features. The subroutines are automatically executed when the operator reaches the fields where they were specified. Application programs may be run concurrently with other DEF functions; application programs can be used for functions such as printing reports received on a DEF disk from a host system or sorting and merging data files. Data entry subroutines and non-data-entry application programs can be written in COBOL or assembly language.

DEF Operations

All system operations, including data entry and system control, are performed from the VIP 7200 operator display station. Data entry and control is initiated from the keyboard. User-generated information (e.g., data entry forms) and system-generated information (e.g., selection lists and prompting messages), and all entered data are displayed on the screen. Selection lists indicate which options may be selected (e.g., data entry, supervisory functions). Prompting messages request entry of specific responses or information.

Data is entered at an operator display station onto a form created for the particular application. As data is entered, specified data validation and editing takes place. If an error exists, an alarm sounds. Error messages are displayed on the bottom line of the operator display station. The appropriate correction can be made and, if desired, additional data entered. The data entered is stored on disk and is available to be processed, viewed, modified, deleted, printed, or transmitted using the GCOS 6 file transmission utility program.

DEF Supervisory Functions

DEF supervisory functions have the capabilities listed below; access to these functions is protected via a password to prevent unauthorized or accidental access to the functions:

- Copy forms, tables, and data files
- Rename forms, tables, and data files
- Delete forms, tables, and data files
- Assign any number of operator display stations to a selected printer or change the current disk volume assignments for forms, tables, and data files.
- Examine and change the system status
- Change the supervisory password

DEF Utilities

Utility routines provide the following capabilities:

- Display (1) form names, (2) names and types of tables (i.e., extract or verify table), or (3) file names, number of records in each file, and each file's verification status
- Print (1) form names, (2) names and types of tables, or (3) file names, number of records in each file, and each file's verification status
- Display names of disk volumes to which the operator display station has access for reading and writing forms, tables, and data files.
- Temporarily change disk volumes that forms, tables, and data entry files are read from or written to for the operator display station at which the user is working.

DEF Configuration

The user must configure the system using configuration directives (see the *System Building* manual). He must link the DEF object modules to form a bound unit. He then creates a task group for DEF and creates DEF-specific tasks.

For a complete description of the Data Entry Facility, refer to the *Data Entry Facility User's Guide* manual.

FILE TRANSMISSION BETWEEN LEVEL 6 AND OTHER COMPUTERS

File transmission between the Level 6 and a variety of other processors (Level 6, 62, 64 and 66, Series 200/2000, and non-Honeywell processors) is implemented through three utility pro-

grams: TRAN, TRANH, and TRANB. Each of these utility programs permits files to be transmitted to or received from one or more remotely located processors. Each processor must incorporate appropriate file transmission software.

The TRAN utility program provides for file transmission between the Level 6 and one or more Level 66 host processors. The TRANH utility program is used for file transmission between the Level 6 and other Level 6 processors, or between the Level 6 and Level 62, Level 64, or Series 2000 host processors. Both TRAN and TRANH transmit files in ASCII format, using the polled VIP protocol.

A third utility program, TRANB, enables file transmission between the Level 6 and non-Honeywell processors that use the BSC 2780 protocol; TRANB converts ASCII data in Level 6 files into EBCDIC 80-character records for transmission, and converts the received EBCDIC records into ASCII format.

Each file transmission program is invoked by a command (either entered on a terminal or included within a user EC command file). The command name corresponds to the name of the utility program invoked: TRAN, TRANH, or TRANB. Each program provides error analysis. For TRAN and TRANH, an initiate/accept dialog between file transmission software in each of the two processors determines whether a file can be transferred. A restart capability is available when transmission between two Level 6 processors or between a Level 6 and a Level 66 processor is aborted due to failure in the transmission line. File transfer can be restarted at any record in the file being transferred at the time of failure.

Multiple file transmissions between the Level 6 and one or more processors can occur concurrently. (For example, the Level 6 could transmit files to a Level 64 and a Level 66 host processor concurrently.) Each file transfer takes place over a different communications line. An argument in the command that invokes the file transmission program specifies whether a specific communications line is to remain connected after a file transfer or is to be disconnected. As long as the line is connected, file transfers can be made by issuing the appropriate command (TRAN, TRANH, or TRANB) for each transfer.

For details on the use of the file transmission utility programs to transmit files to a specific processor, refer to the appropriate file transmission manual.

APPENDIX A

BES/MOD 400 COMPATIBILITY

Many areas of GCOS 6 MOD 400 and GCOS/BES are fully compatible, while others require conversion aids. Fully compatible areas include:

- o Source and object text programs
- o Register usage at execution time
- o Data management functions
- o Data file
- o Most executive functions

This appendix describes the considerations that must be made in order to run BES programs under MOD 400 successfully.

EXECUTING BES EXECUTIVE SYSTEM SERVICES UNDER MOD 400

The MOD 400 Monitor and I/O system services are a superset of the BES online Executive system services. Therefore, the same register interface is used, except:

- o $B2 and $B4 – not available for user values under MOD 400.
- o $B3, $B4 and $B5 – not available for user values under BES.
- o MOD 400 uses a system service macro call to request system services instead of the LNJ instruction used by BES.

There are two methods available to resolve the LNJ macro call and register differences: a Honeywell-supplied Accommodation Package, or user-coded macro calls.

Honeywell-Supplied Accommodation Package

The Honeywell-Supplied Accommodation Package contains the BES external definition (EDEF) system service entry points, and issues appropriate MOD 400 macro calls. It can be loaded as a system extension when CLM initializes a system, or it can be linked as an object unit with the BES object program.

The BES Executive System services handled by the Accommodation Package fall into three groups: those that are completely emulated; those emulated, but with restrictions; and those not emulated at all.

Completely Emulated BES System Services

The following BES Executive system services are completely emulated by the Accommodation Package:

- o Request (ZXRQST)
- o Wait (ZXWAIT)
- o Terminate (ZXTERM)
- o Locate RCT (ZXLRCT)
- o Overlay Loader (ZXOVLY)
- o Operator Output (ZITYP)
- o Operator Reply (ZITYPR)
- o IORB Request (ZIOREQ)
- o BES File Manager (ZYFILE)
- o Error Handlers (ZUERS, ZUERSR)

BES System Services Emulated with Restrictions

The following BES Executive system services are emulated with restrictions:

- o Buffer Manager Calls (ZXBGET and ZXBPUT)
- o Clock Functions (ZXCMGR)
- o Dequeue (ZXDQRB)

The BES Buffer Manager get buffer (ZXBGET) and put buffer (ZXBPUT) calls must be handled by including the Buffer Manager as well as the emulator in the MOD 400 system. The Buffer Manager can either be loaded as a system extension when CLM initializes the system, or an object unit version of it can be linked with the BES object program. The pool parameter table (PPT) is user-created and must reside in the BES program's resulting bound unit, because MOD 400 does not generate one.

The clock (ZXCMGR) "connect," "disconnect," and "millisecond conversion" functions are not handled by the emulator. The "get date/time" function is also not handled by the emulator because it does not use the LNJ interface, but an absolute memory interface (ZXCTOD). However, the (ZXCMGR) "time-of-day ASCII conversion," "date ASCII conversion," and get-internal-time functions are handled by assuming that the user-supplied five-word table contains MOD 400 rather than BES values; therefore, it is important that the program does not manipulate these fields before passing them to the ASCII conversion functions. As with all monitor and I/O system services, it is recommended that these routines be coded as macros under BES (see "User-Coded Conversion" below) so that conversion to MOD 400 will be more easily facilitated by recording the macro rather than attempting to find all of the Monitor and I/O system services in all programs.

Note:

The "connect-with-wait" function is completely emulated.

Althouth the dequeue function (ZXDQRB) is emulated, a subsequent "post" function cannot be issued to the dequeued request block. When the dequeue function is issued a default return code is placed into the referenced request block.

BES System Service Functions Not Emulated

The following functions are *not* emulated.

- o Clock Manager (ZXCMGR) "connect," "disconnect," and "millisecond timer conversion" functions (i.e., function codes 0, 1, and 2, respectively)
- o Clock (ZXCTOD) "ascertain date/time" function
- o Post (ZXPOST)
- o I/O subroutine (ZIOSUB) "common driver" function
- o Operator Attention (ZIATTN)

In addition, permanent clock table blocks are not emulated.

User-Coded Conversion

If program development is done under BES for execution under MOD 400, a macro convention can save overhead (required by the Honeywell-supplied Accommodation Package), although its use necessitates reassembly. The following example illustrates how a task request macro, which contains an LNJ interface, in a BES program can be modified to use a system service macro call interface under MOD 400.

Example:

Assume that the following macro definition appears in a BES program:

```
RTASK  MAC
       LAB   $B4,?PA
       LNJ       $B5,<ZXRQST
       ENDM
```

?PA is a variable that identifies the address of a task request block; this variable is specialized in the BES program as follows:

```
RTASK  ERRMSG  (REQUEST ERROR MESSAGE TASK REQ. BLK.)
```

Under MOD 400, the macro must be redefined as follows, although the instruction specializing the variable parameter need not be altered:

```
RTASK MAC
        $RQTSK ?PA   (MOD 400 MONITOR CALL TO REQUEST TASK)
        ENDM
```

This redefinition of the macro call makes it possible to use the BES object program intact because the $RQTSK macro call generates the proper macro call interface.

EXECUTING BES PROGRAMS UNDER MOD 400

Programs executed under BES can be moved easily to the MOD 400 environment because the two systems are compatible. BES object text programs can be brought (i.e., imported) into the MOD 400 environment for relinking via the IM_PAM command (see the Command manual); once imported, BES object text can be mixed with MOD 400 text. Each member in a BES PAM file is converted into an MOD 400 sequential file when it is imported; conversely MOD 400 sequential files can be unloaded (via the EX_PAM command) to a BES PAM file, in which case each file becomes a member of the PAM file.

However, it is important to note that although BES FORTRAN object text can utilize the MOD 400 FORTRAN run-time package, BES COBOL object text requires the BES COBOL run-time package for execution.

Converting BES Programs to MOD 400

In BES, applications are built entirely by CLM; however, the environment in which MOD 400 applications are executed is constructed in stages.

The MOD 400 operating system is configured/initialized by a bootstrap/CLM sequence, which corresponds roughly to the following BES CLM commands:

SYS
TSA
CLOCK
ADMOD
DEVICE
DEVFILE
IOS
QUIT

Then, using the operator's terminal and optional command file(s), the operator controls system capabilities, which correspond to the following BES CLM commands:

SYS(maximum LRN)
DATE

The command is then used to control online applications; the commands used correspond roughly to the following BES CLM commands:

ADMOD
TASK
ATFILE

After the MOD 400 system is bootstrapped, the MOD 400 CLM is used to build all unchangeable structures within the operating system. The MOD 400 DEVICE directive is used to specify all peripherals to the operating system; although this directive performs the same functions as the BES DEVICE and DEVFILE directives, the BES DEVICE directive allowed a user-written attention routine to be entered, whereas MOD 400 does not support this capability. In MOD 400, dedicated physical levels, symbolic names, and LRNs are chosen for each device; since

all task groups contain these specified LRNs, any task can refer to any peripheral. In addition, symbolic names are used to allow programs to request specific peripherals when a file is opened.

The MOD 400 CLM directive MEMPOOL is used to define memory pool sizes in the available memory above the resident Monitor. A task group is executed within one of the memory pools, each of which is used not only for all code (i.e., bound units local to the task group) but also for control structures for the system to control the task group; these structures include:

- o Task group/LRT/LFT
- o Command processor task
- o Each user task
- o System files (user-in, error-out, FCBs and buffers)
- o Each FCB (from ASSOC)
- o Each intermediate buffer
- o Each bound unit

See the *System Building* manual for information about the sizes of these structures.

The CLM directive LDBU is used to add user bound units to the resident Monitor code; the following Honeywell-supplied components can be added:

- o Accommodation Package
- o BES Buffer Manager

The CLM SYS directive allows the selection of the clock frequency and update interval, the specification of either the SSIP or DSIP simulator, and the setting of the sumber of TSAs. The CLMIN directive changes the CLM input file (only if the device on which the file exists was previously defined in a DEVICE directive).

Using operator commands, the operator controls dynamic system characteristics. This capability makes it possible to change the status of peripheral devices and monitor the status of task groups. The SET_DATE command is used to initialize the internal system clock time. Then, using the SPAWN GROUP or CREATE GROUP commands, a task group, in which the application will run, must be created (i.e., spawned); this can be done via a stored operator command processed automatically at the completion of CLM, or at a later time.

If the application consists of a single bound unit and task, the command processor does not have to be declared as the lead task; in this case, the application can be the lead (and only) task. The highest LRN and LFN values used must also be specified; in BES, you cannot use a LRN used by a system driver. In addition, the base level at which the lead task runs is also specified; the level used by the application cannot duplicate one used by the system (e.g., trap, clock, etc.,) or a driver. For portability and/or future expansion to the configuration, the base level of the lead task should be set a few levels below the last one used by the system.

If CREATE GROUP was used, an EGR (ENTER_GROUP_REQUEST) command is used to activate the lead task of the task group. If the lead task is the command processor, it will read commands to control the construction and step sequencing within that task group.

The commands can be used to construct a multibound unit application as well as control sequencing between steps within a task group. The MOD 400 CREATE TASK command is used to define each task of a task group; it establishes a unique LRN and (optionally) a start address other than the default entry point. In addition, each task created requires a bound unit in which its start address is located; depending on the shareability of the bound unit, this is done as follows:

- o If the bound unit name is stated explicitly (i.e., it is not shareable), memory is allocated within the task group's specified memory pool, and the specified bound unit's root segment is loaded into memory.
- o If the specified bount unit is shareable, the first request for it causes it to be loaded into system-dynamic space; if the unit is already loaded, usage-counts in its bound unit attribute section (BAS) are updated to show multiple users; the BAS is also updated when a task's bound unit is specified by stating that it is the same as another defined task within the same task group bound unit.

The created tasks can be activated initially via the ETR (ENTER TASK REQUEST) command; this command is executed for each task that is to be initially active. If the last ETR is done with -WAIT, then the command processor will become active again only when that task terminates; otherwise, the command processor is active with the application and can be used to retrieve status or terminate the application.

If the application uses non-mass storage peripheral (files), a file management OPEN macro call identifying the pathname of the peripheral must be executed. For compatibility, the same filename used in the BES DEVFILE command must be used in the MOD 400 DEVICE command.

If the application uses mass storage files, the pathname identified in the OPEN macro call may start with a circumflex. (The circumflex is ignored by the BES file manager.) In addition, the ASSOC command is the functional equivalent of the BES ATFILE command and must be used as such.

The following BES CLM commands have no equivalent in MOD 400:

- o ATLRN
- o BUFSPACE
- o EQLRN
- o TRAP

There are no ATLRN or EQLRN commands because each task running under MOD 400 has only one LRN, although a procedure can have two LRNs, they will run asynchronously (i.e., not serially) with each other, and with the different register context. There is no BUFSPACE command because the BES buffer management component can be used by the BES object text program; although the memory area for the buffer pools must be located with the user program and the pool parameter table (PPT) set up by the user. There is no TRAP command because MOD 400 does not allow the user to have direct access to the hardware trap vectors; if this function is needed, a special program, which would conver the MOD 400 trap entry call and structure into the structure used by the MOD 400 program(s) as well as issue system Executive service calls to activate the task group's call mechanism when the specified traps occur, must be written.

The following restrictions exist for programs converted to MOD 400:

1. There is no forward referencing between bound units loaded by the CLM LDBU directive.
2. There is no symbol resolution between bound units; externally referenced (EREF) symbols of a bound unit can be resolved only against symbols externally defined (EDEF) by bound units already loaded by the LDBU command.
3. A task group's bound unit will have no initialization subroutine table (IST) processing; an IST is processed only if loaded by the LDBU directive.
4. The IST routine of the program loaded via the LDBU directive cannot increase the size of the loaded program as in BES; the size can only be preserved or made smaller.
5. Two or more LRNs cannot be given to the same procedure in such a way that a single process will be activated in a serial manner when one of the LRNs is referenced.
6. A user-written program must be used to convert the MOD 400 trap call sequence in the BES type.
7. There is no ATTENTION routine in MOD 400.
8. Privileged instructions that exist in BES programs cannot be executed by programs running in the MOD 400 batch task group.
9. The LEV instruction cannot be used in either online or batch dimensions under MOD 400.

APPENDIX B
PROGRAMMING CONVENVENTIONS

The following programming conventions are provided for designing application programs to interface smoothly with system software.

MODULE AND FILE NAME CONVENTIONS

Program names and load module names that begin with Z are reserved for Honeywell use and should not be used for an application program. System module names are six characters in length; the second character defines the system component. Table B-1 lists the first two characters of each system module name and the system component that it relates to.

The names of files that are processed by program development software (compiler, assembler, and so on), are given a suffix by the particular component doing the processing. Table B-2 lists these suffixes.

TABLE B-1. SYSTEM MODULE NAME PREFIXES

Name Prefix	System Component
ZA	Assembler
ZC	COBOL Compiler
ZE	Editor
ZF	FORTRAN Compiler
ZG	Configuration Load Manager
ZH	Trap Handler
ZI	Input/Output Drivers
ZL	Linker
ZM	Memory Management
ZO	Loader
ZP	Macro Preprocessor
ZQ	Communications
ZR	RPG Compiler
ZS	Sort/Merge
ZU	Utility Routines and Conversion Aids
ZX	Executive
ZY	File, Data and Storage Management
ZZ	Program units internal to File, Data and Storage Management

TABLE B-2. SYSTEM PROGRAM FILE NAME SUFFIXES

Suffix	File Type
.A	Assembly language source unit
.AO	Default user-out if user-in is disk
.C	COBOL language source unit
.EC	Execution command (EC)
.F	FORTRAN language source unit
.L	List unit
.M	Link maps
.O	Object unit
.P	Macro Preprocessor source program unit
.Q	RPG Compiler generated linker directive file
.R	RPG language source unit
.WA	Writable Control Store (WCS) assembly language source unit
.WO	WCS object unit

CALLING SEQUENCE FOR EXTERNAL PROCEDURES

External procedures are those that are assembled or compiled separately from the calling procedure. These procedures may be either *functions*, that is, procedures returning a single value to the caller, or *subroutines*, namely, procedures that alter data contained in an area common to both the procedure and its caller. For example, the FORTRAN mathematical routines (sine, cosine, etc.) are external procedures. When it is necessary to write an assembly language external procedure, use the calling sequence described below for compatibility with code generated by the language processors.

The external procedure calling sequence generated by the CALL statement in assembly language, COBOL, FORTRAN and RPG is of the form:

```
LAB $B7, list
LNJ $B5, <entry
list – Label assigned to the argument list
entry – External label of subroutine's entry point
```

The external procedure should assume that register B5 contains the address of the caller's return point and register B7 points to an argument list having the format shown in Figure B-1.

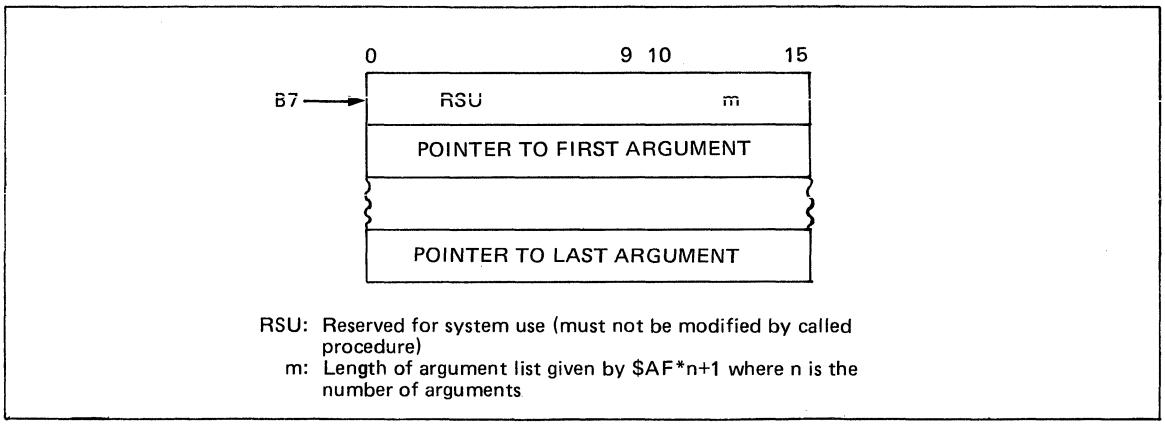


Figure B-1. Argument List

REGISTER CONVENTIONS

The system services uses the following registers *without* preserving their contents: R1, R2, R6, R7, B2, and B4. If the information in these registers is of value to the application program, it should save the register contents before making a Monitor or Input/Output service request. Unless otherwise specified, the following registers will not be altered by the system services: S, I, R3, R4, R5, B1, B3, B5, B6, B7, T, RDBR, CI, SI, S1, S2, S3, and the M registers.

ASSEMBLY LANGUAGE PROGRAM INDEPENDENCE

If an assembly language program is to be run under both SAF and LAF systems, the program must be written using code that is SAF/LAF independent by reassembly, or code that is SAF/LAF independent at loading. The rules for preparing these types of assembly language programs are given in the *Program Preparation* manual.

Self-Modifying Procedures

A self-modifying procedure should not be used for two reasons: (1) it cannot be made reentrant and (2) the instruction as modified might not be executed becasue of the instruction prefetching feature of the model 6/40. In instruction prefetching, an arbitrary number of words are prefetched in parallel with the execution of the current instruction. The prefetch buffer is emptied only when a transfer of control occurs. In case of a store into a word that had previously been prefetched, the prefetch buffer is *not* cleared and the prefetched instruction will be executed as it was prior to modification.

However, if a self modifying procedure must be used, the program must contain code to remove the prefetched instruction after modification is complete but before the modified code is executed. This can be done by executing an unconditional branch of the form:

```
B    $+2    FLUSH THE PREFETCH
```

APPENDIX C
HARDWARE SUPPORTED

HARDWARE RESOURCES

Figure C-1 shows the hardware resources that can be used in a Level 6 configuration. (Minimum configurations are given below.) For a complete description of central processors, peripheral and communications hardware, refer to the Level 6 Minicomputer Handbook.

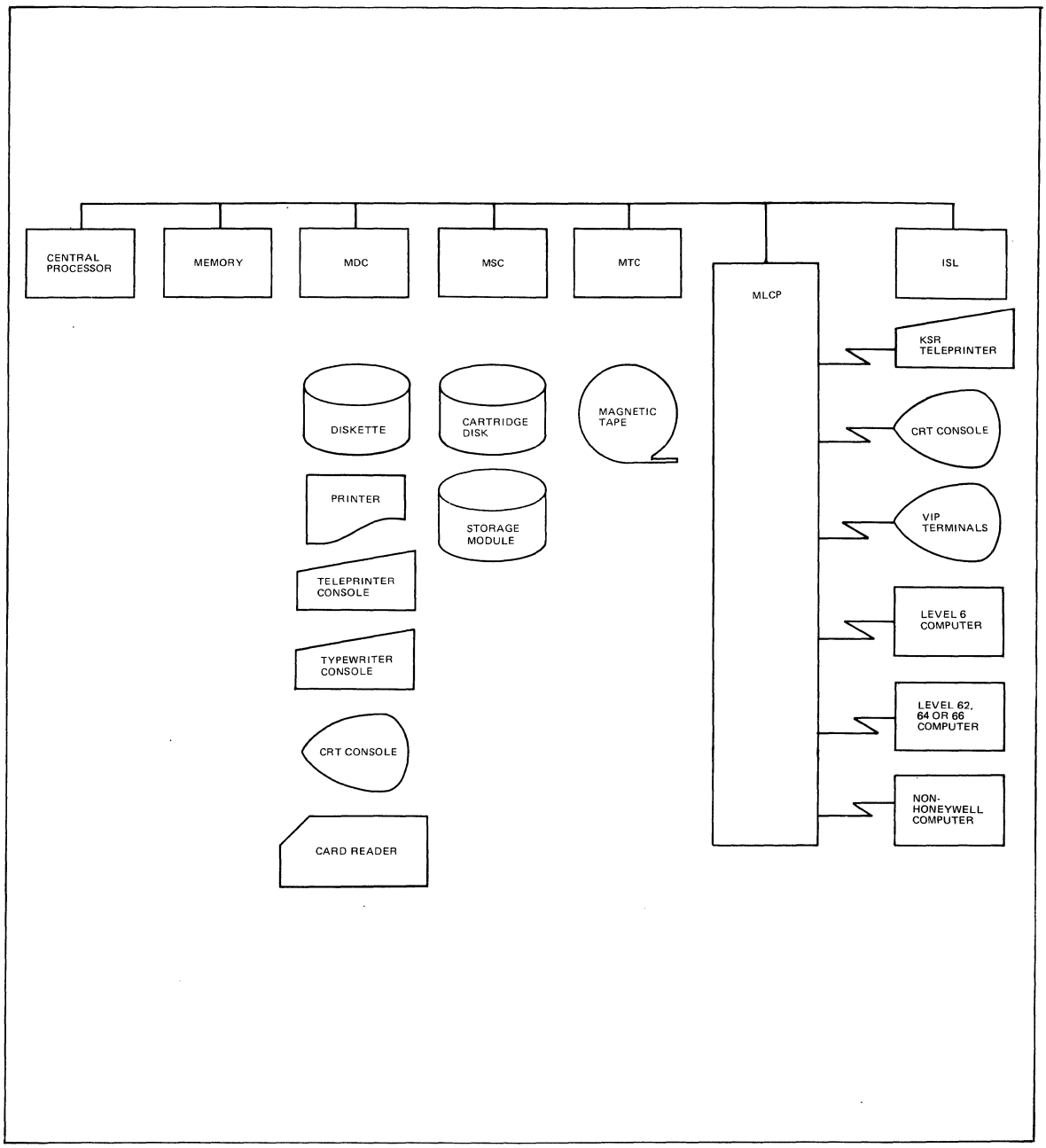


Figure C-1. Level 6 Hardware

Memory is available in multiples of 8K words up to 256K words, depending on the central processor. A multiple device controller (MDC) controls terminals, printers, diskettes, and card readers. A mass storage controller (MSC) controls fixed and removable cartridge disks, and storage modules. A magnetic tape controller (MTC) controls 7- and 9-track magnetic tape devices. The terminals are: keyboard-send-receive (KSR) and automatic-send-receive (ASR) teleprinters (no paper tape); cathode ray tube keyboard console (CRT); and typewriter console.

The multiline communications processor (MLCP) is a programmable communications controller that is programmed to handle supported asynchronous terminals, synchronous terminals, and communications to other computers. The MLCP can be programmed by the user to handle devices not supported by Honeywell-supplied software.

The asynchronous terminals supported are: KSR teleprinters, CRT consoles, and visual information projection (VIP) system terminals with line editing at the keyboard, buffered line transmission, and full cursor control to edit text from the keyboard.

The synchronous terminals supported are: VIP terminals with keyboard, screen, and receive-only printer (ROP) that can be used in a poll/select mode of operation. The VIP is an interactive display terminal that, in addition to line cursor control, provides forms control to display a form on the screen for formatted data entry, and function code keys to transmit a function code to be interpreted by the receiver.

The synchronous BSC2780 communications protocol is used for communicating with other Level 6 computers, Level 66 computers, and non-Honeywell computers.

Figure C-1 also illustrates that data transfers occur over communications lines that are either hard-wired, dedicated, or dial up.

The Intersystem Link (ISL) interconnects two busses to extend the bus, or to share central processors, memory or controllers.

EQUIPMENT REQUIREMENTS

Minimum Equipment for Program Preparation

The following equipment is required for program preparation:

- o 6/34, 6/36, or 6/43 central processor with full or basic control panel.
- o 32K words of memory (SAF mode) or 40K words of memory (LAF mode).
- o Operator's terminal (TTU9101, TTU9102, DKU9101, DKU9102, TWU9101, TWU9104, TWU9106), connected through the multiple device controller (MDC); or ASR33, ASR35, KSR33, VIP7100, VIP7200 connected through the Multiline Communications Processor (MLCP).
- o One million bytes disk storage: two dual diskettes (DIU9102) connected through the MDC; cartridge disk (CDU9101, CDU9102, CDU9103, CDU9104) connected through the mass storage controller (MSC); or storage module (MSU9101, MSU9105, MSU9102, MSU9106) connected through the MSC.

Minimum Equipment for Online Applications

The following equipment is required to run online applications:

- o 6/34, 6/36, or 6/43 central processor with full or basic control panel.
- o 24K words of memory (SAF mode).
- o 32K words of memory (SAF mode) for communications applications.
- o Additional 8K for LAF mode.
- o Operator's terminal (TTU9101, TTU9102, DKU9101, DKU9102, TWU9101, TWU9104, TWU9106) connected through the MDC; or operator's terminal (ASR33, ASR35, KSR33, VIP7100, VIP7200) connected through the MLCP.
- o One half million bytes disk storage: one dual diskette (DIU9102) connected through MDC; cartridge disk (CDU9101, CDU9102, CDU9103, CDU9104) connected through the MSC; or storage module (MSU9101, MSU9105, MSU9102, MSU9106) connected through the MSC.

Hardware Supported

Table C-1 lists the central processor, peripheral, communications equipment, and other equipment that can be used.

Modems supported for asynchronous communications terminals are:

- o 103,[1] 113,[1] 202 type modems

For synchronous communications terminals they are:

- o 201, 203, 208 type modems
  Also supported are:
- o Honeywell modem by-pass for both asynchronous and synchronous terminals
- o Modem types where the connection, disconnection and dataset control settings can be user specified

A modem is not required for a direct connect asynchronous terminal or synchronous terminal with a timing source in the terminal or in the MLCP.

Receive-only printers (ROP) can optionally be used with the VIP7700, VIP7760-2A, or VIP7700R. The ROPs are TermiNets 100, 150, 300, 1200, the ASR/KSR33, or the ASR35.

TABLE C-1. HARDWARE SUPPORTED

Type Number	Description
	Central Processors
CPS945X	6/34; additional memory to 32K words
CPS946X	6/36; additional memory to 64K words
CPS955X	6/43; additional memory to 64K words (SAF mode), or to 256K words (LAF mode)
Memory Options	
CMC9001	Parity Memory (Single Fetch)
CMC9002	EDAC Memory (Single Fetch)
CMC9503	Cache Memory (6/40)
CMC9501	Parity Memory (Double Fetch – 6/40)
CMC9502	EDAC Memory (Double Fetch – 6/40)
Central Processor Options (6/40)	
CPF9501	Memory Management Unit
CPF9502	Commercial Instruction Processor
CPF9503	Scientific Instruction Processor
CPF9509	Writable Control Store
CPF9504	Portable Plug-in Control Panel for 6/40
	Peripherals
Card Readers	
CRU9101	300 cpm, punched card
CRU9102	300 cpm, punched and marked sense card
CRU9103	500 cpm, punched card
CRU9104	500 cpm, punched and marked sense card
CRF9101	51-column option for CRUs 9101, 9102, 9103, 9104

[1] These modems must be equipped with the option to disconnect the data set after a carrier drop of 110 milliseconds.

TABLE C-1 (CONT). HARDWARE SUPPORTED

Type Number	Description
CRU9108	300 cpm, punched card reader
CRU9109	300 cpm, punched and IBM marked sense card reader
CRU9110	300 cpm, punched and HIS marked sense card reader
CRU9111	500 cpm, punched card reader
CRU9112	500 cpm, punched and IBM marked sense card reader
CRU9113	500 cpm, punched and HIS marked sense card reader
Card Readers/Punches	
CCU9101	400 cpm card reader/punch
PCU9101	100- to 400-cpm card punch
Cartridge Disk Drives	
CDU9101	Low density 100 tpi removable disk (1.25 million words)
CDU9102	Low density 100 tpi fixed and removable disks (2.5 million words)
CDU9103	High density 200 tpi removable disk (2.5 million words) CDU9104
CDU9104	High density 200 tpi fixed and removable disks (5.0 million words)
CDU9114	100 tpi, low density, fixed and removable disks (2.5 million words)
CDU9116	200 tpi, high density, fixed and removable disks (5.0 million words)
CDU9115	High Density removable disk (2.5 million words)
Storage Modules	
MSU9101	40 megabyte, 411 cylinders
MSU9105	40 megabyte, 411 cylinders
MSU9102	80 megabyte, 823 cylinders
MSU9103	143/127 megabyte
MSU9104	288/256 megabyte
MSU9106	80 megabyte, 823 cylinders
Diskette Drives	
DIU9101	Single diskette
DIU9102	Dual diskette
Line Printers	
PRU9103	240 lpm, 96-character set
PRU9104	300 lpm, 64-character set
PRU9105	480 lpm, 96-character set
PRU9106	600 lpm, 64-character set
PRU9108	660 lpm, 96-character set
PRU9109	900 lpm, 64-character set
PRF9102	12-channel vertical format unit option for PRUs 9103, 9104, 9105, 9106, 9109
Serial Printers	
PRU9101	60 lpm, 64-character set
PRU9102	60 lpm, 96-character set
PRU9112	120 cps, 96-character set (Lina 21)
PRU9114	160 cps, 96-character set

TABLE C-1 (CONT). HARDWARE SUPPORTED

Type Number	Description
Console Devices	
TTU9101	Teleprinter console (ASR33)
TTU9102	Teleprinter console (KSR33)
DKU9101	CRT/keyboard console, 64-character set
DKU9102	CRT/keyboard console, 96-character set
DKU9103	CRT/keyboard console, 64-character set
DKU9104	CRT/keyboard console (VIP7205), 96-character set
TWU9101	Typewriter console, 30 characters per second, 64-character set
TWU9104	Typewriter console, 30 characters per second, 96-character set
TWU9106	Typewriter console, 120 characters per second, 96-character set
VIP7200	CRT/keyboard console
Magnetic Tape Drives	
MTU9104	Magnetic tape drive (9-track NRZI, 45 ips)
MTU9105	Magnetic tape drive (9-track NRZI, 75 ips)
MTU9109	Magnetic tape drive (9-track NRZI/PE, 45 ips)
MTU9110	Magnetic tape drive (9-track NRZI/PE, 75 ips)
MTU9114	Magnetic tape drive (9-track PE only, 45 ips)
MTU9115	Magnetic tape drive (9-track PE only, 75 ips)
MTU9116	Magnetic tape drive (9-track NRZI, 45 ips)
MTU9117	Magnetic tape drive (9-track NRZI, 75 ips)
MTU9112	Magnetic tape drive (7-track NRZI, 45 ips)
MTU9113	Magnetic tape drive (7-track NRZI, 75 ips)
MTU9120	Magnetic tape drive (7-track NRZI, 45 ips)
MTU9121	Magnetic tape drive (7-track NRZI, 75 ips)
Magnetic Tape Controller	
MTC9101	Magnetic tape controller for NRZI tape drives
MTC9102	Magnetic tape controller for PE/NRZI tape drives
	Communications Equipment
Multiline Communications Processor (MLCP)	
MLC9101	With Communications-Pac for eight asynchronous lines
MLC9102	With Communications-Pac for eight synchronous lines
MLC9103	Multiline Communications Processor only – requires Communications-Pac(s) depending on choice of line speeds.
Communication-Pacs (Communications Adapters)	
DCM9101	Communications-Pac, two asynchronous lines, with cable
DCM9102	Communications-Pac, one asynchronous line, with cable
DCM9103	Communications-Pac, two synchronous lines, with cable
DCM9104	Communications-Pac, one synchronous line, with cable
DCM9110	Communications-Pac – Autocall unit for one or two synchronous or asynchronous lines

Type Number	Description
DCM9106	Communications-Pac, one synchronous HDLC line, with cable (for RBF)
DCM9112	Communications-Pac, broadband HDLC line (Bell 301, 303 compatible to 72 KB)
DCM9113	Communications-Pac, broadband HDLC line (CCITT/V35 compatible to 72 KB)
DCM9115	Communications-Pac, broadband synchronous line, (MIL 188C compatible to 72 KB)
DCM9116	Communications-Pac, dual asynchronous line, (MIL 188C compatible to 9.6 KB)
Asynchronous Terminals	
ASR-33	Keyboard/printer, 110 baud
ASR-35	Keyboard/printer, 110 baud
KSR-33	Keyboard/printer, 110 baud
VIP7100	CRT/keyboard, up to 9600 baud
VIP7200	CRT/keyboard, with cursor control, line editing and buffered transmission, up to 9600 baud
VIP7205	CRT/keyboard/display terminal, 96-character set
TWU1001	Keyboard/printer 30 cps
TWU 1003	Keyboard/printer 30 cps
TWU1005	Keyboard/printer 120 cps
PRU1001	Printer Terminal 30 cps
PRU1003	Printer Terminal 30 cps
PRU1005	Printer Terminal 120 cps
Synchronous Terminals	
VIP7700R	Nonpolled CRT/keyboard with optional ROP; 2000 to 4800 baud
VIP7700R	Polled CRT/keyboard with optional ROP; 2000 to 4800 baud
VIP7760-2A	Polled CRT/keyboard with optional ROP; 2000 to 4800 baud
VIP7700	Nonpolled CRT/keyboard with optional ROP; 2000 to 4800 baud
VIP7700	Polled CRT/keyboard with optional ROP; 2000 to 4800 baud
Receive-only printer for CRT terminals	
TN300	Printer Terminal 30 cps
TN1200	Printer Terminal 120 cps
PRU1001	Printer Terminal 30 cps (available on VIP7100 and VIP7200 only)
PRU1003	Printer Terminal 30 cps
PRU1005	Printer Terminal 120 cps
	General Purpose Hardware
Intersystem Link	
GIS9010	Intersystem Link (ISL)

APPENDIX D

GLOSSARY

abort
: An operator action resulting in the immediate cessation of operation of a task group or the operation of the currently executing request in a task group. All resources are returned to the operating system. The bound unit of the lead task of an aborted request may be retained.

activate
: An operator action resulting in the resumption of a previously suspended task group. See suspend.

active
: A task is in the active state when it is executing or ready to execute, when its priority level becomes the highest active one in the central processor.

address, absolute
: A reference to a storage location that has a fixed displacement from absolute memory location zero.

address, relocatable
: A reference to a storage location that has a fixed displacement from the program origin, but whose displacement from absolute memory location zero depends upon the loading address of the program (see relocation factor).

algorithm
: A set of well-defined rules for the solution of a problem.

application program
: A user-written program for the solution of a business, industrial, or scientific problem.

argument
: User-selected items of data that are passed to a procedure. For example, Monitor macro call arguments that are passed to the called system service, or command arguments passed to the invoked task (see parameter).

ASCII (American Standard Code for Information Interchange)
: The interchange code established as standard by the American Standards Association.

attribute, file
: Any of the set of file characteristics which determine its accessibility and degree of protection from task groups other than its current "owner." Established at file creation, attributes can be modified by a command from the owning task group.

attribute, NONSHARE
: The characteristics of a file which deny access by the batch task group.

attribute, READ
: The file characteristic which permits reading of the file by tasks in a task group.

attribute, SHARE
: The file characteristic which permits batch task group access to a file.

attribute, WRITE

The file characteristic which permits writing to a file by tasks in a task group.

base level

See priority level, base.

batch dimension

An execution environment used primarily for non-real-time activities such as program development.

batch pool

The memory pool from which the batch task group is supplied memory segments. It can be rolled out by a task executing in an extendable online pool.

batch task group

The single task group that executes in the batch dimension; it owns a set of resources: the batch memory pool, and the peripheral devices currently available to it.

BCB

See buffer control block.

block

A logical unit of transfer between main memory and a tape device; the size is variable up to a maximum specified for the file.

Binary Synchronous Communications (BSC)

A communications procedure, using a standardized set of control characters and control character sequences, for the synchronous transmission of binary-coded data.

bootstrap routine

A routine, contained in a single record that is read into memory by a ROM bootstrap loader, which reads the operating system into memory. (See ROM bootstrap loader.)

bound unit

The output of one Linker execution that is placed in one file. A bound unit is an executable program consisting of a root segment and zero or more related overlay segments.

break

A user action, initiated by pressing the break or interrupt key, that interrupts a running task so that commands can be entered, the task can be halted temporarily, or termination of the task can be effected.

breakpoint

The assembly language instruction BRK, or the point in a program where such an instruction is inserted for the purpose of interrupting execution and activating a debugging program.

buffer control block (BCB)

A control structure, contained in the system pool area, which describes the characteristics of the buffer.

buffer, I/O

A storage area used to compensate for the differences in the flow rates of data transmitted between peripheral devices and memory.

BYE

A user action, via command, resulting in the immediate cessation of the currently executing request in the task group issuing the command, and the return to the system of all resources except the lead task's bound unit. (See abort.)

byte
A sequence of eight consecutive binary digits operated upon as a unit.

calling sequence
A standard code sequence by which system services or external procedures are invoked.

CCP
See channel control program.

channel control program (CCP)
A program that resides in the MLCP and processes characters, protocol headers, and framing characters.

CI
See control interval.

CIP
Commerical Instruction Processor. A hardware option available on 6/40 models that executes a set of business-oriented instructions.

CIP Simulator
A software component that provides the same functionality as the CIP.

clock frequency
The line frequency, in cycles per second, that is the basis (coupled with the scan cycle) for calculating the interval between real time clock-generated interrupts.

Clock Manager
A Monitor component that handles all requests to control tasks based on real-time considerations, and requests for the time-of-day and date in ASCII format.

clock request block
A control structure supplied by a task to request a service from the Clock Manager.

clock scan cycle
The time in milliseconds between clock-generated interrupts.

clock timer block
The control structure used by the Clock Manager to control the clock-related processing of tasks.

command
An order that is processed by the command processor.

command input file (command-in)
Any file or device from which commands to the command processor are read.

command language
The set of commands that can be issued by a user to control the execution of the user's online or batch task.

command level
The state of the command processor, when it is capable of accepting commands, indicated by the display of the RDY (ready) message.

command processor
A software component that interprets control commands issued by the operator or a user, and invokes the required function.

communications device
A device that transfers data over communications lines and is connected through the MLCP.

concurrency

The read or write file access that the reserving task group intends for its tasks and the read or write file access that the reserving task group allows to other task groups.

configuration

The procedure that involves the use of configuration directives to define a system that corresponds to actual installation hardware.

control interval (CI)

A logical unit of transfer between main memory and a disk device; the size is specified by the user and remains constant for a file. The CI determines the buffer size.

CTB

See clock times block.

CRB

See clock request block.

device driver

A software component that controls all data transfers to or from a peripheral or communications device.

device-pac

The adapter between an MSC or MDC controller and peripheral device (e.g., printer, diskette drive).

direct access

The method for reading or writing a record in a file by supplying its key value.

directive

A "secondary" level order read through the user-in file to a "secondary" processor. Examples are Editor, Linker, Patch, Debug, and CLM (configuration) directives.

directory

A structure in a volume directory containing a description of a file or another directory.

disk

A generic name for mass storage devices such as diskette, cartridge disk, and storage module.

dormant state

A task is in the dormant state when there is no current request for the task.

entry point

A symbolic start address within the root segment of a bound unit.

equal name convention

A special pathname convention that can be used with certain commands to automatically construct the output pathname entry name when the input pathname entry name has been resolved.

error output file (error-out)

The file or device by which the system communicates error information to the user or operator; established when a group request is entered.

exclusive online pool

A memory pool whose boundaries do not overlap those of other pools.

expandable online pool

An online pool that may expand into the batch pool space.

extent
A group of contiguous allocated sectors on a disk.

external procedure
A routine that is assembled or compiled separately from the program that calls it.

FCB
See file control block.

FDB
See file description block.

FIB
See file information block.

field
A collection of meaningful data consisting of bit patterns that can be translated into alphabetic, numeric, and special characters in a standard character set.

file
A collection of one or more records.

file control block (FCB)
A File System data structure that controls a user's access to a file. A FCB is pointed to by an entry in the logical file table, and in turn, points to a file description block. There is one FCB per user LFN associated with a file.

file description block (FDB)
A File system data structure that describes a file or directory. A FDB is pointed to by a FCB for a particular file. There is one FDB per file or directory concurrently known (reserved) in the file system.

file information block
A user-created data structure containing required information for file processing.

file name
A 1- to 12-character name assigned to a collection of related data records, or to a peripheral or communications device. For a file on disk this name is assigned when the file is created. For devices, the name is assigned at system configuration. See pathname.

file organization
A method that establishes a relationship between a record and its location in a file. See fixed relative, indexed, relative, or sequential file organization.

File System
The operating system software that handles input/output functions of each of the supported input/output devices.

fixed-length record
A record stored in a file in which all of the records are the same length.

fixed relative file organization
A relative file organization that is compatible with the BES Executive system.

floatable overlay
An overlay that can be loaded into any available memory location within a task group's memory pool.

function
A procedure that returns a single value to its caller. Compare with subroutine.

group id

See task group identification.

HMA

High memory address. The address of the highest physical memory location in the central processor.

IMA

See immediate memory addressing.

immediate memory addressing

A form of addressing a location in main memory by referencing the location directly, indirectly, or through direct or indirect indexing.

indexed file organization

A disk file whose records are organized to be accessed sequentially in key sequence or directly by key value.

input/output device

A peripheral or communications device.

input/output request block (IORB)

A control structure used for communication between a program and an I/O driver outside of the file system.

interrupt

The initiation, by hardware, of a routine intended to respond to an external (device-originated) or internal (software-originated) event that is either unrelated, or asynchronous with, the executing program.

interrupt save area

An area used to store the context of an interrupted task. There is one ISA for each task in memory.

interrupt vector

A pointer to a priority-level-specific memory area called an interrupt save area. There is one vector for each priority level, each having a dedicated memory location.

Intersystem Link

A hardware element interconnecting two busses, thereby permitting the same functions between two units on different busses as between two units on the same bus.

IORB

See input/output request block.

ISA

See interrupt save area.

ISL

See Intersystem Link.

KSR

A keyboard send-receive teleprinter.

KSR-like terminal

A KSR teleprinter, CRT keyboard, or VIP terminal, which supports the TTY protocol, either connected to the MDC, or MLCP.

lead task

The controlling task of a task group. The lead task can invoke other tasks to perform functions on its behalf (i.e., Monitor or I/O services).

LFN
See logical file number.

LFT
See logical file table.

line
A record stored in a Series 60-compatible file.

line number
The relative position of a record in a line, in a block or control interval in a Series 60-compatible file.

line protocol handler (LPH)
A communications program that processes messages, interrupts, and timeouts; handles protocol acknowledgement and error recovery; initializes the channel control program.

loader
A Monitor component that dynamically loads from disk the root and overlays of a bound unit from disk.

logical file number (LFN)
An internal identifier that becomes associated with a file when it is reserved. LFNs are used in all file references until the file is removed.

logical file table (LFT)
A data structure for use by the File System. It contains an entry for each logical file number.

logical resource number (LRN)
An internal identifier used to refer to tasks or devices.

logical resource table (LRT)
A data structure within a task group containing an entry for each logical resource number used in an application, or a data structure within the system task group containing an entry for each logical resource number representing a device. Each entry is a pointer to the resource control table (RCT).

login
A procedure used to gain access to the system. When the login procedure has been executed, it spawns a task group to be associated with the user's terminal.

LPH
See line protocol handler.

LRN
See logical resource number.

LRT
See logical resource table.

MBZ
"Must be zero."

MDC
Multiple device controller for peripheral devices other than cartridge disk, storage module, and magnetic tape.

memory pool
A block of central processor memory from which a task group obtains segments of memory as required for executable code, control structures and input/output buffers. See batch, online, or system pool.

Memory Manager

A Monitor component that controls dynamic requests for memory or to return memory to a memory pool.

MLCP

See multiline communications processor.

MMO

Memory management option. A hardware feature, only on the Model 6/40, which provides memory management options.

MSC

Mass storage controller for cartridge disks or storage modules.

MTC

Magnetic tape controller for magnetic tapes.

multiline communications processor (MLCP)

A programmable interface between a central processor and one or more communications devices. Can be programmed to handle specific communications devices.

multiprogramming

An operating system capability that allows the concurrent execution of tasks from more than one task group.

multitasking

An operating system capability that allows the concurrent execution of more than one task in one or more task groups.

NATSAP

Next available trap save area pointer.

nonfloatable overlay

An overlay that is loaded into the same memory location relative to the root each time that it is loaded.

nonexclusive online pool

A memory pool whose boundaries can overlap those of other nonexclusive online pools.

object unit

A relocatable program unit produced by a single execution of the FORTRAN, RPG or COBOL Compiler, or by the Assembler, and requiring further processing by the Linker to produce a bound unit.

online dimension

An execution environment intended for use by application programs, including those operating in real time.

online pool

A memory pool from which an online task group is supplied memory segments. An online pool can be shared by more than one task group. See exclusive, nonexclusive or expandable online pools.

online task group

A task group that executes in the online dimension; its resources are an online memory pool and the peripheral devices it requests.

operating system area

The memory area containing operating system software, user-written extensions to the operating system, and device drivers.

operator commands

The set of commands that can be issued by the system operator to control execution in the online and batch dimensions.

Operator Interface Manager

A Monitor component that manages all messages sent simultaneously by multiple task groups to the operator terminal or from the operator terminal to a task group.

operator output file (operator-out)

The file or device by which an interactive command communicates with the system operator; established at system initialization or when a FILE_OUT command is issued.

operator terminal

A KSR-like terminal specified for use in interactive communications between the operator and Honeywell-supplied and user-written application programs.

overlay segment

A section of a program that can be loaded during execution to overlay another section of the program. Used when there is insufficient memory to accommodate all the code of a program. See floatable overlay and nonfloatable overlay.

parameter

The data received by a procedure that is written in a generalized form to handle any data passed to it (see argument).

patch

A portion of code used to modify an existing object or load unit on disk or in memory.

pathname

A character string by which a file, directory or device is known in the file system.

pathname, absolute

A pathname that begins with a greater-than sign (>), or a circumflex (^). In the former case, it is a partial pathname, and is appended to the root directory name of the system volume to form a full pathname; in the latter case, it is a full pathname, and is used without modification.

pathname, device

A pathname by which reference is made to a peripheral device via the symbolic peripheral device (SPD) directory. Device pathnames have the general form >SPD>dev_name.

pathname, relative

A pathname that does not begin with a greater-than sign (>), or a circumflex (^). It is a partial pathname consisting of one or more directory names and/or a file name, and is appended to the working directory pathname to form a full pathname.

pathname, simple

A special form of a relative pathname consisting of a single directory name or file name. It is appended to the working directory name to form the full pathname.

peripheral device

A device connected through the MDC, MSC or MTC (e.g., a card reader, disk or tape).

physical input/output

Physical input/output, or physical I/O, which is initiated through a request I/O macro call, outside of the file system.

PIO

See physical input/output.

pool identifier

A two-character name, established at system configuration, by which a memory pool is identified, and by which a task group is assigned a memory pool when the task group is created.

priority level

A numeric value that may be assigned to a task or device for purposes of controlling processing. Values range from 0 to 63. The lowest values (highest priorities) are reserved for system tasks; level 63 is the system idle level. Intermediate levels are available for user assignment to tasks and devices. The physical level at which a task executes is the sum of the highest system physical priority level plus one, the base level of the task group, and the relative level of the task within the group.

priority level activity indicators

Each bit of a 4-word-area in memory is used to indicate whether a task is active at that level.

priority level, base

The priority level, relative to the system priority level, at which all tasks in a task group execute. A base level of 0 is the next higher level above the last (highest) system priority level.

priority level, physical

See priority level.

priority level, relative

The priority level, relative to the base level, at which a user task within a task group executes. Relative level 0 is the base level.

priority level, system

The priority level assigned to system devices and tasks.

program name suffixes

A "point-letter" character string such as ".O" for object units, ".A" for assembly language source units, appended to a file name to identify it as a source object, or list unit.

range

The number of bytes transferred during an I/O operation.

record

A collection of logically related fields.

RCT

See resource control table.

reentrant routine

A routine that during execution does not alter itself; a reentrant routine can be entered and reused at any time, by any number of callers.

relative file organization

A file whose records are organized to be accessed sequentially or directly by their record position relative to the beginning of the file.

relative level

See priority level, relative.

relative record number

A number representing the position of a record relative to the beginning of the file. The initial record is relative record number 1.

request block
See IORB, TRB, CRB, SRB.

request I/O
The macro call, issued to a driver, that performs physical input/output (PIO).

request queue
A threaded list of request blocks.

resource control table
A control structure created for use by the Monitor to control task processing.

resident bound unit
A bound unit that is permanently configured in memory as an extension to the operating system.

return address
The address of the instruction in a program to which control is returned after a call to a subroutine. By convention, this address is usually stored in register B5.

RFU
"Reserved for future use."

residual range
The difference between the number of bytes requested and the number of bytes transferred during an I/O operation.

ROM bootstrap loader
A firmware routine (activated by pushing the Load key on the control panel) that reads the first record from a designated disk into memory.

root directory
The base of the directory structure on a disk volume; it has the same name as the volume name recorded on a disk volume. It is identified by a preceding circumflex (^).

root segment
The controlling segment of a program. It is resident in memory during the entire execution of the program, and can call overlay segments.

RSU
"Reserved for system use."

search rules
An ordered list of directories that are searched by the operating system when a bound unit is to be located and loaded or executed.

sector
A 128-byte portion of a diskette track, or a 256-byte portion of a cartridge disk or storage module track.

semaphore
A software counter mechanism, available to assembly language programs, and used to coordinate the use of task code or other resources such as files.

semaphore request block (SRB)
A data structure used to control semaphore processing.

sequential access
The method of reading or writing a record in a file by requesting the next record in sequence.

sequential file organization
A file on disk or magnetic tape whose records are organized to be accessed in consecutive order.

shareable bound unit
A transient bound unit consisting of reentrant code and residing in the system memory pool. It is available for execution as a task of more than one task group.

shareable file
Any file that is usable by more than one task concurrently.

SIP
Scientific Instruction Processor. A hardware option on 6/40 models that executes a set of scientific instructions.

SIP Simulator
A software component that provides the same functionality as the SIP.

source unit
A program written in source language for processing by a compiler or an assembler. Source units are stored as variable sequential data files.

spanned record
A variable-length record that is segmented and spans one or more control intervals; valid only for disk-resident sequential files.

spooling
The technique for storing output on disk files for subsequent printing.

SRB
See semaphore request block.

standard I/O files
The command-in, user-in, user-out, operator-out, and error-out files.

star name convention
A special pathname convention that can be used with certain commands to perform an operation on a group of files, thereby eliminating the need for separate commands for each file.

startup
The procedure that bootstraps a Honeywell-supplied, preconfigured system from disk, to provide a minimum operating environment.

states (task)
A task can be in the following states: dormant, active, wait, and suspend.

subroutine
Any procedure that alters data in an area common to both the subroutine and its caller. Contrast with "function".

suspend
An operator action resulting in the temporary cessation of execution of a task group; all resources are retained by the task group. See activate.

system console
See operator terminal.

system task group
The task group in which all drivers, the clock, the command processor and OIM execute.

system pool

The memory area from which the system task group and system global structures (e.g., BCB and FDB) are supplied with memory segments, and the area where shareable bound units reside.

system directory

One of the directories that the operating system uses in its search for a bound unit to be loaded for execution.

task

A sequence of instructions that has a starting point, an ending point, and performs some identifiable function.

task control block

The system control structure that describes the task's characteristics, including the contents of the hardware interrupt save area (ISA).

task group

A named set of one or more tasks which has a common set of resources; the framework within which every user and system function operates.

task group identification

A two-character name by which a task group is known to the operating system.

task group resource

One of a set of elements associated with a task group which enables it to perform its function. A resource can be a task, a central processor priority level, central processor memory, or a peripheral or communications device.

Task Manager

A Monitor component that handles task requests to activate, wait, or terminate tasks.

task request block (TRB)

A data structure used by one task to request another task and communicate with it.

TCB

See task control block.

terminate

A system service macro call request issued by the currently executing task at the end of its normal processing.

thrashing

The situation describing the frequent roll-out/roll-in of the batch pool.

transient bound unit

A bound unit that resides in memory as long as there is a request for it.

transparent mode transmission

A data transmission mode that allows data consisting of bytes having any bit configuration to be transmitted over communications lines. Thus, control characters can be transmitted as data.

trap

A control transfer caused by an executing program. The transfer is made to a predefined location in response to an event that occurs during processing.

trap handler

A routine designed to take a particular action in response to a specific trap condition.

Trap Manager

A Monitor component that handles an executing program's transfer of execution control to a predefined trap location.

trap save area

An area in memory in which certain information is stored when a trap occurs.

trap vector

A pointer to a trap handler. There is one vector for each possible trap condition, in dedicated memory locations.

TRB

See task request block.

TSA

See trap save area.

user identification

A field that identifies the current user of a task group.

user input file (user-in)

The file or device from which a command function requiring directives (e.g., the Editor) reads its input; it is established when the group request is made. User programs can also read from this file.

user output file (user-out)

The file or device by which an interactive command communicates with the user; established when a group request is made, or a FILE_OUT (FO) command is issued. User programs can also write to this file.

variable-length record

A record stored in a file in which records have different lengths.

volume header

A control structure on every disk or magnetic tape volume that carries information about the volume.

volume name

See root directory.

wait

A task is in the wait state when it causes its own execution to be interrupted until a timer request is satisfied, until another task releases a semaphore, until another task terminates, or until an I/O operation terminates.

WCS

See Writable Control Store.

word

A sequence of 16 consecutive binary digits operated upon as a unit; two consecutive bytes.

working directory

A directory pathname associated with a task group. It begins with a root directory name and contains zero or more intermediate directory names. It is used by the File System software to construct a full pathname whenever a task group refers to a relative or simple pathname.

Writable Control Store

User programmable firmware.

CUT ALONG LINE

HONEYWELL INFORMATION SYSTEMS

Technical Publications Remarks Form

TITLE: SERIES 60 (LEVEL 6) GCOS 6 MOD 400 SYSTEM CONCEPTS

ORDER NO. CB20, REV. 0

DATED JANUARY 1978

ERRORS IN PUBLICATION

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

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

FROM: NAME _____ DATE _____

TITLE _____

COMPANY _____

ADDRESS _____

_____

CUT ALONG LINE

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

FOLD ALONG LINE

FIRST CLASS
PERMIT NO. 39531
WALTHAM, MA
02154

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

Postage Will Be Paid By:

HONEYWELL INFORMATION SYSTEMS
200 SMITH STREET
WALTHAM, MA 02154

ATTENTION: PUBLICATIONS, MS 486

FOLD ALONG LINE

Honeywell

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.

19804, 3178, Printed in U.S.A.

CB20, Rev. 0