Honeywell

SOFTWARE OVERVIEW AND SYSTEM CONVENTIONS

SERIES 60 (LEVEL 6)

GCOS/BES2

SOFTWARE

Honeywell

SOFTWARE OVERVIEW AND SYSTEM CONVENTIONS

SERIES 60 (LEVEL 6)

GCOS/BES2

SUBJECT:

Introduction to Series 60 (Level 6) GCOS/Basic Executive System 2 Software.

SOFTWARE SUPPORTED:

This manual supports Release 0200 of Series 60 (Level 6) GCOS/Basic Executive System (BES2) software. See the Subject Directory for a list of other publications that support this release.

DATE:

July 1976

ORDER NUMBER:

AU50, Rev. 0

PREFACE

This manual introduces the Series 60 (Level 6) GCOS/Basic Executive System 2 (BES2) software provided for the Series 60 (Level 6) 6/30 models hardware. Unless stated otherwise herein, the term BES will be used to refer to the GCOS/BES2 software; the term Level 6 will indicate the specific models of Series 60 (Level 6) on which the described software executes.

Section 1 presents a brief introduction to the configurable hardware, hardware features, and software components. It contains descriptions of related BES documents.

Section 2 describes, in more detail, system software components.

Section 3 describes the general considerations involved in testing and executing application programs.

Section 4 specifies the various system conventions relating to data and file organizations and formats, character set and conversion tables, programming and error reporting.

Appendix A is a list of the system modules as they are distributed, along with their names and approximate sizes.

Appendix B shows minimum equipment requirements and indicates additional usable equipment.

Appendix C is a glossary of BES terms.

File No.: 1S13

GCOS/BES2 SUBJECT DIRECTORY

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

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

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

Subject	Order No.
Replace Memory Values (Utility Set 1)	AU47
Scientific Branch Simulator	AU45
Software Release Materials (Contents)	AU50
System Conventions	AU50
System Software and Documentation (Overview)	AU50
Task Manager	AU45
Trace Trap Handler	AU45
Transfer Input to Disk File/Member (Utility Set 2)	AU47
Trap Handling (Offline)	AU46
Trap Handling (Online)	AU45
Utility Programs	AU47

The following publications constitute the Series 60 (Level 6) GCOS/BES2 manual set. They support Release 0200 of Series 60 (Level 6) GCOS/BES2 and fully describe its major components.

Order Number	Manual Title	Revision	Date
AS32	Series 60 (Level 6) GCOS/BES FORTRAN Reference Manual	1	July 1976
AU41	Series 60 (Level 6) GCOS/BES2 COBOL Reference Manual	0	July 1976
AU43	Series 60 (Level 6) GCOS/BES2 Assembly Language Reference Manual	0	July 1976
AU44	Series 60 (Level 6) GCOS/BES2 BASIC Reference Manual	0	July 1976
AU45	Series 60 (Level 6) GCOS/BES2 Executive and Input/Output	0	July 1976
AU46	Series 60 (Level 6) GCOS/BES2 Operator's Guide	0	July 1976
AU47	Series 60 (Level 6) GCOS/BES2 Utility Programs	0	July 1976
AU48	Series 60 (Level 6) GCOS/BES2 Program Development Tools	0	July 1976
AU49	Series 60 (Level 6) GCOS/BES2 Planning and Building an Online Application	0	July 1976
AU50	Series 60 (Level 6) GCOS/BES2 Software Overview and System Conventions	0	July 1976

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

Order Number	Manual Title	Revision	Date
AS22	Honeywell Level 6 Minicomputer Handbook	0	January 1976

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

Order Number	Manual Title	Revision	Date
AT97	Series 60 (Level 6) MLCP Programmer's Reference Manual	0	June 1976

CONTENTS

CONTENTS (cont)

ILLUSTRATIONS

ILLUSTRATIONS (cont)

SECTION 1
INTRODUCTION

The Basic Executive System (BES) consists of system software that supports the development of your application programs, and the execution of your application programs within minimum hardware configurations.

The objectives of BES are to provide a system that:

- Supplies programs and utilities to assist application program development
- Supports real time operations
- Provides communications support
- Is time and interrupt driven
- Allows device independent programming
- Supports program overlay capability
- Handles multiple user tasks
- Provides configurable memory-resident executive software

A variety of system programs provide you with the means to develop your application and configure your execution environment. BES supplies:

- Four programming languages
- Program development tools
- Utility programs
- Memory configuration program

The modular design of BES execution time software enables you to design and select the minimal operating environment suited to your application. BES provides:

- Peripheral and communications device support
- Executive control routines
- File management routines
- Input/output routines

HARDWARE

Figure 1-1 shows the hardware resources that can be used in a Level 6 configuration. (A minimum configuration is given in Appendix B.) For a complete description of central processor, peripheral and communications device hardware, refer to the Minicomputer Handbook.

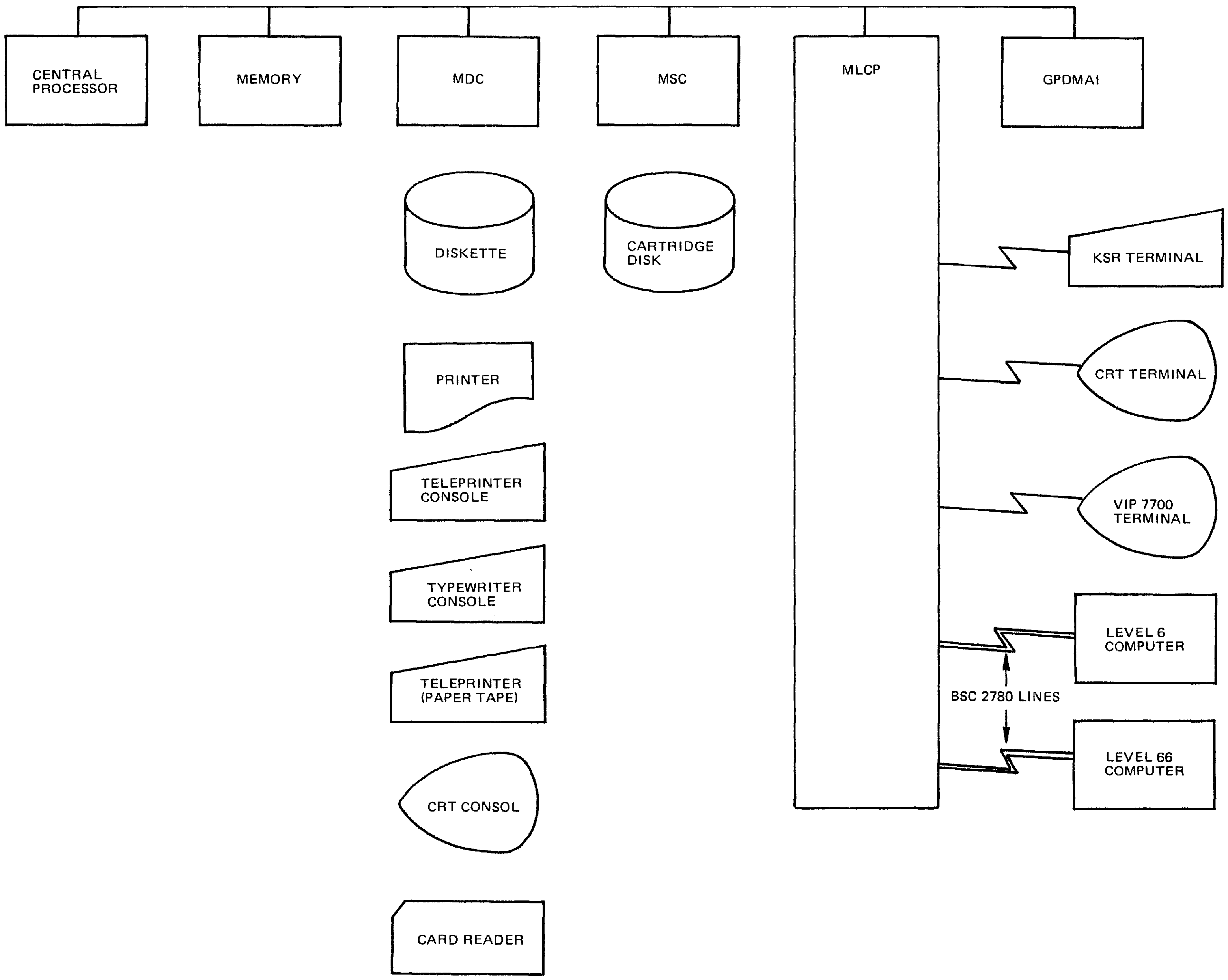


Figure 1-1. Level 6 Hardware

Memory is available in multiples of 8K words up to 64K words. A Multiple Device Controller (MDC) controls consoles, printers, diskettes, paper tape teleprinters, and card readers; a Mass Storage Controller (MSC) controls cartridge disks. The consoles are the KSR (keyboard send-receive) and ASR (automatic send-receive when not using paper tape) teleprinters, CRT (cathode ray tube) keyboard console, and typewriter console.

The Multiline Communications Processor (MLCP) is a user-programmable communications controller that handles asynchronous KSR-like terminals, synchronous Visual Information Projection (VIP) System 7700 terminals, and for communicating with other Level 6 or Level 66 communications computers, it handles lines that use the Binary Synchronous Communications (BSC) 2780 communications protocol transmitting at rates up to 9600 bits per second. The KSR-like terminals are the KSR and CRT terminals. The figure also illustrates that data transfers occur over communication lines either hard-wired, dedicated, or dial-up. The VIP 7700 is an interactive display terminal with line cursor control to edit text prior to transmission; 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, e.g., attention function.

The general purpose direct memory access interface (GPDMAI) is a user-programmable controller that allows the attachment of non-Honeywell peripheral devices. It requires a user-written driver.

FIRMWARE/HARDWARE FEATURES

BES software is supported by firmware and hardware features that contribute to overall system efficiency by reducing memory space occupied by software and by increasing the speed of certain operations above what it would be if they were managed by software alone. Some of these features are described below.

Sixty four interrupt priority levels are provided, of which the highest ones are reserved for specific system functions, the lowest one is the "system idle" level and all others are available for assignment to devices and program tasks. 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; a task can initiate another task for execution or terminates itself by calling task management functions. Multiple tasks operate idenpendent of and asynchronous to each other. The task at the highest active interrupt priority level is the one in execution. If in order for the higher priority level task to execute, a task at a lower priority level must be interrupted, the context of the lower priority task is saved (e.g., the hardware registers and program counter). When the task at the higher priority level terminates, the context of the task at the lower priority level is restored, and it executes. Interrupting 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.

Another feature is the trap handling function. This function causes an immediate control transfer to the designated software routine upon the detection of conditions such as program errors, hardware errors, arithmetic overflow, and uninstalled optional instructions.

OPERATING ENVIRONMENTS

BES system software provides two operating environments; the offline environment primarily for program development, and the online environment for application program execution. Figure 1-2 identifies the environments in which system software operates, and shows individual BES modules grouped by function.

Offline Environment

The offline software consists of independent system programs that are used to develop your application programs and to prepare your files. These system programs operate in an environment designed to efficiently process a series of system programs, one program at a time.

Briefly, the available program development tools are: the Command Processor, to sequence the execution of requested system programs; the Editor, to create and correct a source program, written in one of the provided programming languages and stored in a source program module; the language processors, to create a relocatable object module from a source program module; the Linker, to produce an executable load module by combining one or more object modules; and the Cross-Reference Program, to produce a cross reference list of all identifiers used in an assembly language program.

The language processors are: a FORTRAN Compiler that supports the proposed 1976 American National Standards Institute FORTRAN subset, a COBOL Compiler patterned after the low-level 1974 American National Standard COBOL, an expanded BASIC Interpreter, and a Macro Preprocessor and Assembler for the BES assembly language. Note, the BASIC Interpreter is a self contained subsystem that enables you to prepare and execute a BASIC program without using the Editor, Linker or Executive.

OFFLINE ENVIRONMENT

PROGRAM DEVELOPMENT TOOLS

- Command Processor
- Editor
- Macro Preprocessor
- Assembler
- FORTRAN Compiler
- COBOL Compiler
- BASIC Interpreter
- Linker
- Cross-Reference Program

UTILITY PROGRAMS

- Utility Set 1
  initialize volume/file,
  allocate, delete, list,
  dump, replace, rename
- Utility Set 2
  print, dump logical file,
  dump disk area,
  disk to paper tape,
  card/paper tape to disk
- Utility Set 3
  copy, compare
- Bootstrap Generator
- Dump Edit
- Offline Debugger
- Program Patch

INPUT/OUTPUT DRIVERS

- Card Reader
- Printer(s)
- KSR Console
- ASR Teleprinter
- Diskette
- Cartridge Disk

OTHER SOFTWARE

- MLCP Macro Routines

ONLINE ENVIRONMENT

EXECUTIVE MODULES

- Task Manager
- Clock Manager
- Operator Interface Manager
- Buffer Manager
- Overlay Loader

INPUT/OUTPUT MODULES

- File Manager
- FORTRAN Run-Time I/O Routines (FRIOR)
- COBOL Run-Time I/O Routines
- Drivers
  Card Reader
  Printer(s)
  KSR Console
  ASR Teleprinter
  Diskette
  Cartridge Disk
- Communications Drivers
  KSR Terminal
  VIP 7700 Terminal
  BSC 2780

OTHER SOFTWARE

- Online Debug Program
- Trace Trap Handler
- FORTRAN Run-Time Routines
- Floating-Point Simulator
- Scientific Branch Simulator

ONLINE/OFFLINE

- Disk Loader
- Paper Tape Loader
- Card Loader
- MLCP Loader
- Configuration Load Manager

Figure 1-2. BES Software

BASIC is an easy to use programming language designed for the entry level programmer; FORTRAN is intended for scientific application programming and COBOL for business application programming.

To facilitate program development and application execution a full set of utilities is provided for preparing and maintaining disk volumes and files, transferring data between media, copying and verifying disk data, debugging application programs, and patching object or load modules on disk. Two disk devices are supported, diskette and cartridge disk.

Input/output drivers are supplied to transfer data between a program and a peripheral device supported in the offline environment.

The MLCP macro routines can be used to interface new communications equipment. They are used by system programmers to write the channel control programs, which control the flow of data between the central processor and communications devices.

Online Environment

Application programs are executed in an online environment that you configure. The principle characteristic of the online environment is that is can respond to interrupts from external devices, internal hardware (e.g., the real-time clock), or software, and execute the program task at the highest active interrupt priority level. The system software controlling this environment is contained in routines that the application program must explicitly call, if it wants to exercise multitask control. The software modules special to the online environment will now be described briefly.

The Executive modules are: the Task Manager, to schedule and synchronize tasks including other executive and input/output system tasks; the Clock Manager, to service real-time clock interrupts and provide task scheduling after timed intervals; the Operator Interface Manager, to control dialogue between an application program and system operator; the Buffer Manager, to control the dynamic allocation of preselected blocks of memory to an application program; the Overlay Loader to load, during execution, application memory image segments to a designated portion of memory.

Physical I/O drivers are provided for all peripheral and communications devices to transfer data between memory and devices. These include interactive consoles that are connected to the MDC, and interactive terminals that are connected to the MLCP to transfer data over communications lines. To further simplify I/O programming, File Manager routines, that perform logical I/O operations on files, are provided for all devices. Both the I/O drivers and the File Manager are implemented as reentrant routines. The FORTRAN and

COBOL Run-Time Input/Output Routines use the File Manager to support file I/O and data formatting.

The Online Debug Program and Trace Trap Handler are available for testing programs in the online environment.

FORTRAN Run-Time Routines are mathematical routines, bit string manipulation routines, and routines to implement task management capabilities.

The Floating-Point Simulator and the Scientific Branch Simulator provide software simulation for scientific instructions.

Loaders are provided to load executable load modules from peripheral devices into memory. The Configuration Load Manager uses these loaders when it configures the online environment by loading the application load modules, building Executive data structures, and initiating program execution. The MLCP Loader loads the channel control programs into the MLCP.

COMMUNICATIONS

Communications software provides you with easy to use interfaces to KSR, CRT, VIP 7700, and BSC 2780 communications devices, identical to those provided for peripheral devices. Code written for communications can be used to access peripheral devices, enabling you to select I/O devices for production runs and for program checkout. The BSC 2780 is treated by the software like other devices and is considered to transfer unbuffered files in either direction, but in a nonconversational mode.

SOFTWARE DOCUMENT SET

This overview manual is intended to introduce you to BES software. Most of the information presented here is of a general nature, and contains references to the more detailed treatment of particular subjects in the other manuals in the set.

Some of the information in this overview is not to be found in any other manual in the set. The material in Section 4 on media and system conventions, and that in Appendix A about system software modules is unique to the overview.

Briefly, the contents of the other documents in the set are:

- Honeywell Level 6 Minicomputer Handbook – Descriptions of hardware models, central processor, processor architecture and features, instruction set, registers, peripheral devices, software, various controllers and system features, as well as maintenance and site preparation information.
- Assembly Language Reference manual – Complete description of all instructions, instruction formats, control statements, types of addressing, types of data handled, and macro language statements.

- FORTRAN Reference manual – Complete description of all statements, instruction formats, types of files and data handled. FORTRAN run-time support routines (intrinsic functions, tasking, I/O).
- COBOL Reference manual – Complete description of the general features of COBOL programs, language elements, language syntax, the four major divisions of a COBOL program, specific format descriptions of all COBOL statements (including programming examples incorporating each statement), and the types of files and data handled.
- BASIC Reference manual – Complete description of all BASIC statements, statement formats, mathematical functions and operating procedures. Includes a description of the loading procedure.
- Program Development Tools manual – Detailed descriptions of the Command Processor, Editor, Linker, Cross-Reference Program; formats and arguments for all commands used for these programs, diagrams and examples of use of the development tools. Includes a description of the loading procedure and optional functions of the Assembler, FORTRAN and COBOL Compilers, and Macro Preprocessor.
- Utility Programs manual – Detailed descriptions of each utility set and its functions, the commands and operands associated with its use; examples of each function and sample printouts. Contains a separate list of all operands used by the utility programs, a definition of the operands, and a list of associated functions. Contains descriptions of the Offline Debugger, Program Patch, Dump Edit, and Bootstrap Generator.
- Executive and Input/Output manual – Complete descriptions of all Executive modules: the Task, Clock Buffer, Operator Interface Managers, and Overlay Loader, as well as File Manager and Communications; all online and offline device drivers; the Trace Trap Handler, Floating-Point Simulator, and Scientific Branch Simulator.
- Operator's Guide – Descriptions and examples of all control panel and system console procedures required by an operator to start up a system, load programs and execute them, display values on the control panel, change these values. This manual includes descriptions of the Disk, Paper Tape, and Card Loaders, and a complete list of error messages issued by all system software.
- Planning and Building an Online Application – Description of the elements involved in designing an online application. Details the actions of the Configuration Load Manager (CLM) in specializing and combining Executive modules to satisfy user requirements; defines all CLM commands and parameters. Describes the Online Debug Program.

SECTION 2
SYSTEM SOFTWARE COMPONENTS

Before an application program can be executed, it must be written using the program development tools provided by the offline environment. The assembly language programmer must know the online functions that are available if task scheduling or file I/O is used in the program, since these online functions are called directly by assembly language code. A COBOL or FORTRAN program must execute in the online environment, but all interfaces with online functions are provided in language statements; no direct calls to these functions are made.

This section describes features of the offline and online components.

PROGRAM DEVELOPMENT TOOLS

The program development tools are the language processors, Editor, and Linker system programs which are executed in the offline environment. Another system program, the Command Processor, controls the sequencing and loading of these programs. The program development tools require a minimum configuration consisting of a console, two diskette drives or a cartridge disk, and 16K words of memory.

Program Development Sequence

Figure 2-1 shows the program development sequence for application programs other than BASIC, which is described in the BASIC language manual. All disk volumes are assumed to have been initialized and files allocated prior to the development sequence shown in the figure.

To explain the sequence, the following paragraphs describe the preparation of an assembly language application program. You can use a KSR command input device to enter commands to the Command Processor that it prepare the Editor for execution. After each command, control returns to the KSR until you request to load the Editor. The Command Processor activates the Disk Loader to load the Editor. You interact with the Editor to enter source language statements. The Editor builds a source module and stores it on disk. When the Editor terminates, control returns to the Command Processor (not illustrated) and you can request the Assembler for execution.

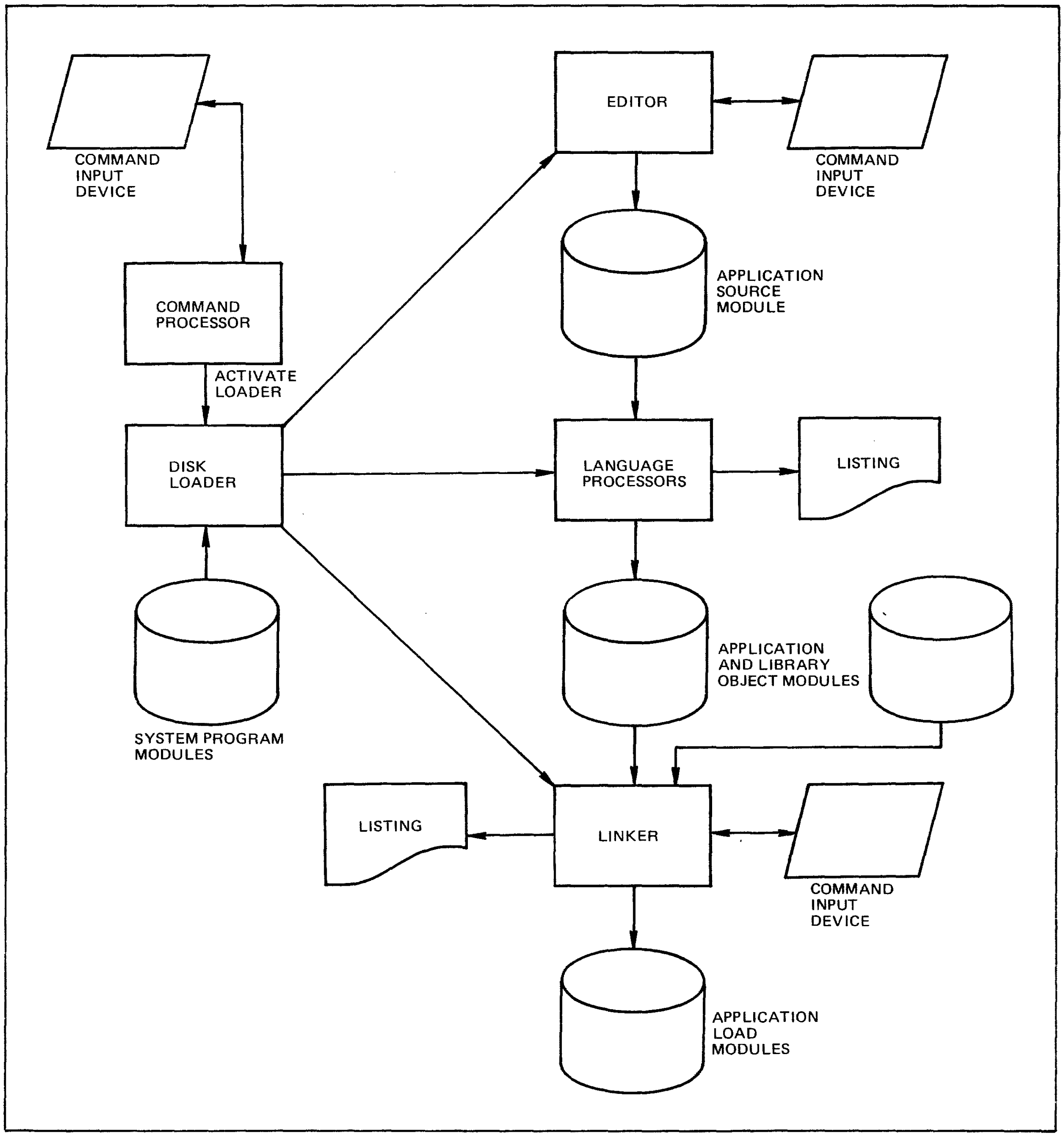


Figure 2-1. Application Program Development Sequence

To run the Assembler, enter the appropriate commands (similar to those used for the Editor) to the Command Processor. In particular, the source module file is identified. The Assembler is loaded, it reads the source module from disk, and assembles the program to create an object module. If the program is in error, you must correct the source module, using the Editor, before processing can be continued.

After all the programs to be executed are assembled, the Linker is loaded and links the object modules, from one or more disks, into a load module, which is suitable for loading and execution.

In summary, the procedure used for obtaining system program functions is to enter commands to the Command Processor to prepare the specified system program for execution, until you request that it be loaded. The system program runs and performs its function. Then control returns to the Command Processor, and you can request another system program.

The above example used an interactive KSR as the command input device. Other devices, such as CRT, card reader, or disk can be used. However, the latter two provide command input files and cannot be used interactively. For example, in a card reader command input file to the Editor, the cards will contain Editor commands followed by other cards with additional source statements or corrections to statements in the source file.

Included in the "language processor" function of Figure 2-1 is the Macro Preprocessor which is required to process an assembly language application source module containing macro calls. Such a source module must be processed by the Macro Preprocessor, which creates on disk another application source module with assembly language source code replacing the macro calls. The latter source module is the new input module to the Assembler.

The following paragraphs describe the program development components.

<u>Command Processor</u>

The Command Processor is provided to sequence the execution of program development system programs in the offline environment.

Before a system program can be executed, commands must be issued to the Command Processor describing the files to be used by the system program, files such as the program file on which the system program resides (e.g., the file containing the Assembler); the data input file to the system program (e.g., assembly language source file); and the output file from the system program (e.g., assembled object module). Another command loads the system program and initiates execution. This command also is used to pass arguments to the system program; e.g., "the suppress all listings" or "list errors only" arguments to a language processor. The Command Processor accepts typeins from a console, and command files from the disk or card reader.

A description of the Command Processor commands and those used with each system program (except the Basic Interpreter) are contained in the Program Development Tools manual.

Editor

The Editor is a system program that enables you to create and/or correct the source text of macro routines or assembly language, FORTRAN, or COBOL programs.

Editor accepts its source text input from disk and writes the corrected source text output to disk; it can take the command file that directs its operation from a console, card reader, or disk.

Editor can also be used to add a new source program to disk from punched cards or console typeins.

See the Program Development Tools manual for details of all the Editor commands for locating, substituting, deleting, and inserting statements in source programs.

Once your application program has been written, and necessary editing operations have been completed, it must be processed by one of the following language processors.

Macro Preprocessor

The Macro Preprocessor provides a convenient method for including in an assembly language source module a specified sequence of source statements. This capability simplifies the coding of source modules and enables you to standardize the coding of frequently used statement sequences. Each of these statement sequences is contained in a macro routine which can be created and stored on disk macro libraries by the Editor, or is located at the beginning of the assembly language source module.

Wherever the sequence of statements is to be included in the source program, a macro call is written. Prior to assembly, a source module containing macro calls must be processed by the Macro Preprocessor to substitute a corresponding sequence of source statements for each macro call.

The Macro Preprocessor requires a minimum of 16K words of memory, a disk that may be used for both input and output, and a console.

A description of the Macro Preprocessor commands is contained in the Assembly Language manual.

Assembler

The Assembler is a system program that processes assembly language source code, translates these statements into object code acceptable to the Linker, and produces a listing of the source statements and their associated machine code equivalents, suitably flagged for error conditions.

The Assembler is a two-pass program that runs in a minimum configuration of 16K. The first pass generates the symbol table in the Assembler's resident table area; the second pass generates the object module and/or a listing, as requested.

The Assembler is invoked by a command entered through a console. It accepts source statement input from disk. Assembler output, both the object module and the listing, may be written to disk. Alternatively, the listing may be assigned directly to a printer.

The code for analyzing the options passed to the Assembler, as well as that required for initializing the symbol table, is overwritten by the operation of the Assembler. For this reason, only one source program can be assembled for each load of the Assembler.

A description of the assembly language instructions is contained in the Assembly Language manual.

FORTRAN Compiler

The FORTRAN Compiler translates FORTRAN source statements into an object module ready for processing by the Linker, or optionally, into a source module of assembly language statements for processing by the Assembler. The compiler is a single-pass processor that operates in a minimum of 16K words of memory.

The compiler is invoked by a command entered through a console; it accepts its source input from disk and produces its object module or assembly language source module on disk. It also produces a listing of the source text (with imbedded diagnostics and a memory map) that is directed either to a disk for printing later, or to a printer.

Routines for data conversion and input/output (sequential and direct access), as well as routines for FORTRAN intrinsic functions are available in object text format to be linked, as needed, to FORTRAN programs. Additionally, FORTRAN offers the following Instrument Society of America (ISA) extensions: bit string manipulation on values of integer data, and seven subroutines to implement task management capabilities.

A description of the FORTRAN language statements and run-time routines is contained in the FORTRAN manual.

COBOL Compiler

The COBOL Compiler translates COBOL source statements into an object module ready for processing by the Linker. The compiler is a multipass processor that operates in a minimum of 16K words of memory.

The compiler is invoked by a command entered through a console; it accepts its source input from disk, and produces its object module on disk. It also produces a listing of the source text (with imbedded diagnostics and a memory map) that is directed either to a disk for printing later, or to a printer.

Additionally, COBOL offers the following: support of relative files and random access capability, the CALL statement, and debug lines.

A description of the COBOL language statements is contained in the COBOL manual.

BASIC Interpreter

The BASIC Interpreter provides interactive facilities to create, modify, store, retrieve, and execute programs written in the BASIC language. An important part of the Interpreter's program exectuion function is its ability to process disk data files that have been prepared by either BES BASIC or BES FORTRAN programs. It operates in 16K words of memory.

By fully utilizing the program and file processing facilities available in the BASIC Interpreter, you can operate within a self contained BASIC-oriented environment for most of the more elementary data processing and problem-solving requirements and applications. Both program libraries and data files can be created and maintained on disk for this purpose.

The Interpreter is invoked by a command entered through a console; input is from a console or disk. Diagnostics and execution results are displayed on the console.

A description of BASIC program preparation, execution initiation, and language statements is contained in the BASIC manual.

Linker

The Linker combines object modules that are the output of a compiler or the Assembler, and converts them to a format acceptable for loading. It resolves external references, and can process one or more object modules to produce a single load module, or several load modules, in one execution. For planned overlays it produces a root load module plus overlay modules.

The Linker is controlled by command statements that are entered either through a console, card reader, or from a member of a disk file. It accepts the object modules it processes from disk; it writes its output (load modules) to a disk.

Linker also produces listings containing a link map and error messages that can be written to a console, a disk, or a printer.

A description of the Linker commands is contained in the Program Development Tools manual.

Cross-Reference Program

The Cross-Reference Program can be used to produce an alphabetic list of all symbolic names (i.e., labels and identifiers) in an assembly language source module. The program lists for each symbolic name its line references in the source module, and it flags an undefined symbol or a multiply-defined label. Input to the Cross-Reference Program is in source module form and from disk. Output is a listing to a printer or console.

A description of the Cross-Reference Program operating instructions is contained in the Program Development Tools manual.

UTILITY PROGRAMS

The system programs designated as utilities provide a variety of services including disk volume preparation and maintenance, file handling, data transfer from one type of medium to another, and program patching. These programs execute only in the offline environment, and are loaded from disk using the Command Processor.

Many of the utility functions are concerned with files on disk. As explained in Section 4 of this manual, two file types are supported: relative and partitioned. Relative files contain fixed-length records that can be accessed either sequentially or directly. Partitioned files, used only by BES system programs, contain variable-size members such as source modules, object modules, list modules, or load modules. Utility functions handle relative files with deletable records or multiple extents, but cannot create files with these properties.

Brief descriptions of the utilities follow; for complete details, see the Utility Programs manual.

Utility Set 1

Utility Set 1 performs disk volume and file preparation and maintenance. One of its functions is to initialize new disk volumes for use by system file handling routines. Volume initialization consists of writing volume directory information on track 0 of the volume. (See Section 4 for the information and formatting provided by this program.)

Other functions are:

- Initialize a disk volume
- Allocate space for new files
- Initialize partitioned files
- Delete files, or members
- Rename a volume, a file, or a member
- List the name, location, or size of all files, freespace and defective sectors of a disk
- Dump from a disk to a memory area, or from memory to: disk, KSR, or printer. Replace one area of memory with a specified value before writing to a device.

Utility Set 2

Utility Set 2 provides data transfer functions. These are:

- Print contents of a file or member containing format control characters
- Dump contents of a file or member on a logical record basis
- Dump contents of a disk on a sector basis
- Transfer a member from disk to paper tape
- Transfer cards or paper tape to disk member

Utility Set 3

Utility Set 3 provides for copying and verifying of disk data. It will:

- Copy a volume, file or member from one disk to another
- Compare the new volume, file, or member with the original to verify the accuracy of the copy operation

Dump Edit

Dump Edit prints from a diskette a memory dump previously dumped there.

Bootstrap Generator

Bootstrap Generator creates a bootstrap record containing parameters to be subsequently used during bootstrapping and loading.

Offline Debugger

The Offline Debugger is an interactive, offline utility used for program testing and temporary error correction. The console dialogue consists of commands submitted by the operator, and responses displayed on a console in the form of informational and error messages. Debugger can display memory and register contents, and allows the operator to modify them. The "breakpoint" feature causes activation of Debugger during the execution of your program at the location where a breakpoint is set.

Program Patch

The Program Patch utility allows the alteration of either object or load module text. Patches may be created, added, deleted, or listed by using the appropriate commands to the utility.

MLCP SOFTWARE

The Honeywell-supplied software support for the Multiline Communications Processor consists of the MLCP Loader and the MLCP macro routines.

MLCP Loader

The MLCP Loader resides in main memory linked to a user-written program, and is responsible for loading the channel control programs and control structures into the random access memory (RAM) portion of the MLCP, for the purpose of processing the communications data stream of an application.

MLCP Macro Routines

There are a number of macro routines available for writing the channel control programs and other control structures for handling communications data streams. These macro routines are processed by the Macro Preprocessor prior to assembly. For a detailed description of the MLCP software, refer to the MLCP Programmer's Reference Manual.

EXECUTIVE MODULES

A set of Executive functions is provided to enable your application program to schedule the execution of its tasks. There are three reasons for wanting to control the execution of tasks: (1) to obtain better utilization of the hardware and, consequently, faster execution; (2) to be able to respond to a real-time event that must interrupt the currently executing task; and (3) to enable tasks to contend for control of the central processor.

Better hardware utilization is possible by running system resources simultaneously, since the system resources, namely, the central processor, real-time clock, peripheral devices, and communications devices operate independently of each other. In your application program, every resource that is to run simultaneously can be associated with a different task and the tasks scheduled for simultaneous execution. For example, the function of one task of an application can be to perform input/output, and that of another, to use the real-time clock functions.

For real-time applications, events can be associated with tasks. A different task can be requested to process the external interrupt from each unique device or the internal interrupt from the real-time clock. Each different task can be associated with a different priority level, and the task at the highest active priority level will be executed. The Planning and Building an Online Application manual lists the priority levels recommended for system resources.

An application, consisting of several tasks, might require that tasks contend for the central processor and that one task be executed prior to or instead of another. For example, a task whose function is monitoring the flow of a liquid can be programmed to always suspend from execution a task whose function is file update. This is done by associating the monitoring task with a higher priority level than the file update task. Then even when both tasks are active, the monitoring task executes.

The functions provided by the Executive components are implemented as subroutines that an assembly language application program must call. (The BES FORTRAN includes subroutines that provide tasking functions.) For example, to request that a task be scheduled, the assembly program must code a call to the Task Manager's task request routine, passing to that routine the entry point of the task.

The Executive components provide the tools for execution control. The degree of execution control that exists during execution is determined by the sequence of the requests to the Task Manager to schedule tasks plus the priority level at which each task executes. The following paragraphs describe the functions of the Executive components.

A description of the Executive module functions is contained in the Executive and Input/Output manual.

Task Manager

Task Manager functions are used by an application program to schedule its tasks for future execution, dispatch the next one for immediate execution, and synchronize the execution of two tasks, where one is waiting for the completion of the other.

A task can use the Task Manager to request that another specified task be activated. During the request or later in execution, the requesting task can ask that it be put in a wait state, until the called task completes execution. This enables the requesting task to synchronize its execution with the completion of the called task. When a task completes, it asks to be terminated. Other functions are described in the Executive and Input/Output manual.

Clock Manager

Clock Manager functions are used by an application program to initiate tasks based on the passage of time.

A task can use the Clock Manager to connect to a clock-timer routine in order to initiate another task, either once or cyclically, after a specified period of time. If it does not want a requested time-out to occur, the task can disconnect the request. A task can suspend its own execution for a specified period of time; it can request the time-of-day and date in ASCII format.

Overlay Loader

The Overlay Loader can be used during execution, by a program whose size is greater than the total available memory space, to overlay specified areas of memory with different application code.

Code loaded at configuration time can call the Overlay Loader during execution, to load an overlay mamber. Specified in the call is the overlay number that each overlay member was given during linking, a relocation address needed for a relocatable overlay, and the start address of the overlay. Additional overlays can be loaded in any order by a call from a loaded overlay or from the originally loaded code.

Operator Interface Manager

Operator Interface Manager functions can be used by an application program to control dialog between itself and the operator's console (i.e., a console or KSR-like terminal). The operator's console displays messages to the operator and accepts operator responses. You can write your own attention character processing routine to supplement the one provided.

A program can request that a message be typed at the console for information to the operator. A different request is possible that types a message at the console, prompting a response from the operator. Outstanding requests are queued, but can be answered in any order. Messages that the operator has not responded to can be recalled and displayed. The output to a screen display can be slowed down.

Buffer Manager

Functions of the optional Buffer Manager can be used by an application program to manage memory that is to be used for work space. During online configuration you can divide work space memory into pools, each containing memory blocks of a particular size.

A program can use the Buffer Manager to get a block of memory for work space by specifying the size of the block; when it is no longer needed the block can be returned to the pool.

INPUT/OUTPUT MODULES

The BES modules that provide data input and output services include a File Manager (logical file and data access functions), FORTRAN and COBOL Run-Time Input/Output Routines (FORTRAN data formatting, and input/output for FORTRAN and COBOL object programs), and device drivers (physical read/write and error handling for all available devices). File Manager works at the program's logical level with files and records; I/O drivers work at the hardware physical level. See the Planning and Building an Online Application manual for further discussion of these differences.

The interface to communications can be through an I/O device driver or File Manager. For both interfaces, the BSC 2780 line protocol is handled by system software. Your application program does not have to include any protocol handling code.

A description of the input/output module functions is contained in the Executive and Input/Output manual.

File Manager

File Manager functions are used by an application program to manipulate files on peripheral or communications devices. File Manager provides the following features:

- Supports all device types including communications
- Allows double as well as single buffering
- Provides for blocking and deblocking of logical records
- Locates the requested disk file by name (thus, the program need not know the physical location of its data)

- Supports disk relative files with fixed length records; each record in the file is uniquely identified by a nonnegative integer which specifies the record's ordinal position in the file
- Supports relative files with deletable records
- Supports both sequential and direct access to a relative file
- Allows you to create a relative file during execution without previously allocating the disk space (referred to as a dynamic file)
- Permits extension of a dynamic file during execution
- Supports dynamic files with multiple extents
- Allows a dynamic file to be a temporary disk work file which is deleted when the file is closed

A task can use the File Manager to open a file. When it is a communications file, opening the file includes a logical line connect. Depending on the device type, a task can read or write logical records. A disk relative file can be rewound to the beginning of the file, or positioned to the end of data; it can also be backspaced or forward spaced. When no further access to a file is desired, the file is closed. A task, during execution, can create and later delete a relative disk file.

FORTRAN Run-Time I/O Routines (FRIOR)

These reentrant routines provide for data transfer, device manipulation, and the processing of data as specified in FORTRAN FORMAT statements. FRIOR is a highly modular package, and only those routines required by a particular FORTRAN program are included when that program is linked. The FRIOR routines use the File Manager 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. Diagnostic messages are produced to inform the user of inappropriate or inconsistent input/output statements.

COBOL Run-Time I/O Routine

This routine provides a logical I/O interface for the transfer and processing of data at object program execution time. The routine is linked with the object program and uses the File Manager to open, close, and position files, and to read and write records to peripheral or communication devices. Diagnostic messages are produced to inform the user of inappropriate or inconsistent input/output statements.

Device Drivers

The device drivers are device-specific components that perform all data transfers between system and application programs and their respective input/output devices. There are drivers for all Honeywell-supplied I/O devices.

Separate, but functionally equivalent, versions of drivers for peripheral devices are provided to operate in the online and offline environments.

In the online environment, device drivers interact with the Execute modules, and receive requests for service via the Task Manager. They use device interrupts to signal the termination of data transfers and special "attention" conditions. The online drivers are reentrant programs capable of supporting the concurrent operation of several devices of the same type. When processing a device interrupt or an input/output service request, the driver runs at the priority level assigned to the particular device being addressed. Online drivers are designed to provide fully simultaneous operation of the Level 6 central processors with multiple input/output operations.

Assembly language application programs can call the device drivers directly, or can use them indirectly by calling the File Manager. (COBOL and FORTRAN object programs use the File Manager.) All drivers have similar calling sequences, and use a standard format input/output request block for communication with the calling program.

You are required to use the logical connect function prior to requesting other communications I/O functions; the disconnect function provides a logical disconnect. However, the system software will ignore the connect/disconnect function requests if the application program uses a peripheral instead of a communications device. (The File Manager and COBOL OPEN/CLOSE implicitly provide the connect/disconnect functions.)

ONLINE DISKETTE DRIVER

The Online Diskette Driver provides simultaneous data transfers to diskettes on separate controllers or device-pacs. Although a device-pac can do only one data transfer at a time, the driver queues a second data transfer request; a data transfer to a second diskette device on the same device-pac appears simultanelus to the application program. Seek operations are always overlapped with program execution.

A task can use the Diskette Driver to read a specified number of bytes from the beginning of a sector into a program-specified buffer. Similarly, a buffer containing a specified number of bytes can be written to the beginning of a sector on diskette.

ONLINE CARTRIDGE DISK DRIVER

The properties and functions of the online Cartridge Disk Driver are similar to those of the online Diskette Driver.

ONLINE PRINTER DRIVER

The online Printer Driver supports both the line printer and serial printer. A task can use the Printer Driver to write a buffer to the printer. It handles variable-length output records and program-supplied vertical format control parameters.

ONLINE CARD READER DRIVER

The Card Reader Driver translates punched card input into ASCII bytes (ASCII mode) or column binary into binary words (verbatim mode). It reads and reports the end-of-file card. A task can use the Card Reader Driver to read a card into memory.

ONLINE KSR DRIVER

The online KSR Driver supports the consoles. It provides "delete-character" and "delete-line" input editing, and reports lengthy device inactivity after input is begun but not completed. It handles program-supplied vertical format control parameters similar to those of the line printer. End of a keyboard transfer is indicated by a carriage return or by exceeding a fixed character count. A task can use the KSR Driver to read keyboard input or to write to the printer or display.

KSR TERMINAL DRIVER

The KSR Terminal Driver is similar in function to the online KSR Driver. In addition, before using a KSR-like terminal over communications lines, a logical line connection must be made. When the line is no longer needed, you can logically disconnect or optionally, hang up the phone.

ONLINE ASR DRIVER

The online ASR Driver, for the peripheral ASR device, supports ASR paper tape operations in addition to keyboard and printer KSR operations. It reads and punches paper tape in 7-bit ASCII or 8-bit binary mode; it allows characters that control the paper tape reader to be transferred by preceding them with an escape character. A program can use the ASR Driver to read or write paper tape, and write an end-of-file mark.

VIP 7700 TERMINAL DRIVER

The VIP Driver supports the VIP Communications device keyboard and screen with the same functionality available in the KSR Terminal Driver.

BSC 2780 DRIVER

The BSC Driver supports half duplex communications lines using the BSC 2780 protocol. It handles program to program transmission of ASCII or EBCDIC character data between a Level 6 and the Level 66 Remote Job Entry subsystem. Additionally, between two Level 6 systems, file transmission of transparent EBCDIC data or any bit pattern is possible (i.e., it allows bit patterns that control the BSC line to be transmitted, by preceding them with an escape character).

A program can use the BSC Driver to logically connect the BSC line. It can read or write up to two system-buffered transmission records. When transmission is completed, the line is logically disconnected.

OFFLINE DEVICE DRIVERS

Offline device drivers, used in BES system programs that operate in the offline environment, are provided for all supported noncommunications devices. These drivers provide functions similar to those of the online drivers. However, they are not reentrant and do not use interrupts, but they do provide a limited capability to overlap processing with I/O activities.

ONLINE DEBUG PROGRAM

The Online Debug Program is an interactive program that executes under Executive control, and is used for application program testing and memory code modification. It uses the operator's console (i.e., a console or KSR-like terminal) for command input and display output. In one configuration, the Online Debug Program is loaded in planned overlays, thereby conserving memory space; in another version it must be memory resident. It is loaded during configuration together with the program tasks being tested. A debug command activates a specified level, and the task executing at that level can be tested.

A programmer at the operator's console enters commands for immediate execution, or for storage on disk for later execution. Breakpoints can be set to trap at selected task code locations. At breakpoints, memory and register values can be displayed and changed. In this way, a task can be executed, the values of its variables checked as execution proceeds, code modified, and if necessary, variable values changed in order to test the sequence of code up to the next breakpoint.

A description of the Online Debug Program commands is contained in the Planning and Building an Online Application manual.

TRAP HANDLERS

A trap is a control transfer made to a predefined location in response to some event that occurs during program execution. Unlike interrupts, which are responses to events that are either unrelated to, or at least asynchronous with, the currently executing program, trap conditions are caused by the executing program.

Level 6 hardware can be enabled to recognize and trap many classes of conditions arising during program execution. Some of these classes are:

- Monitor call (with the MCL instruction)
- Trace (with the BRK instruction)
- Scientific instruction simulation
- Integer arithmetic overflow
- Unprivileged use of a privileged operation
- Reference to unavailable resources
- Program logic error
- Noncorrectable memory error (parity)

See the Executive and Input/Output manual for trap handling details.

BES software provides trap handling facilities for the Trace Trap Handler, the Floating-Point Simulator, and the Scientific Branch Simulator. All other occurrences that result in a trap, cause a halt in processing unless supported by a user-written routine.

Briefly, the Trace Trap Handler maintains a history of specific system parameters such as the program counter (P-register), the system status register (S-register), memory location contents, data and address registers for each "break trap" instruction used in the program.

The Floating-Point Simulator provides 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).

FORTRAN Run-Time Routines

BES software includes a large set of FORTRAN mathematical and bit string manipulation routines. These intrinsic functions are available in object module format, so that they can be linked on an as-needed basis to perform a variety of operations on behalf of a FORTRAN program. Some of the operations performed by these routines are:

- Conversion to and from integer and real values
- Truncation
- Determining the nearest whole number
- Transferring a sign
- Choosing: the largest value; the smallest value
- Finding: the length of a character entity; the square root; the natural logarithm; the common logarithm
- Compute selected plane and sperical trigonometric functions
- Bit string manipulation operations on integer data: inclusive OR, exclusive OR, product, 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:

- Build task control blocks
- Initiate a task after a designated period of time
- Suspend a task
- Return a task control block

See the FORTRAN manual for details about these routines.

CONFIGURATION LOAD MANAGER

After program development is completed, the Configuration Load Manager is used to load and initialize an online application and start execution. It uses supplied information to define system characteristics, and to build the data structures that the Executive software uses to control the processing of tasks.

The commands accepted by CLM are those that set up data structures for the task manager, file manager, devices, trap handling, communications, as well as for the clock variables, and a list of the load modules to be included in the complete system.

The action of the CLM takes place in two phases: the configuration phase, and the loading phase. During configuration, the system data structures are created and stored in main memory, and the load list is created for use in the next phase.

During the loading phase, the various Executive and application modules are brought into memory. Each module contains permanently resident code, and may contain some temporary code for initialization of the module. The temporary code, if any, is executed immediately and then overwritten by the permanet code of the next loaded module. Symbolic references from one load module to another are resolved during the loading phase.

If a program is to use overlays during execution, the action of CLM during the loading phase is to bring each overlay module into the area of memory that it will occupy during execution, and then write it onto a disk file in a fast-loadable form.

When all modules have been loaded and the overlay file, if any, has been written, control is given to the highest active priority level, and execution begins.

LOADERS

Each input device has a loader associated with it to load executable programs (load modules) from the device into memory, and to turn control over to a program.

Programs can be loaded from disk, paper tape (ASR), or card reader. Initially, a read-only memory (ROM) bootstrap loader brings the loader in from the device, and starts it. System programs used for program development are always loaded by the Disk Loader, operating in conjunction with the Command Processor. All loaders can be used with CLM to load online programs.

The loader reads the load module control information, and relocates and loads the code into memory. It also "backpatches" some types of forward and external references that could not be resolved by the Linker. When loading is completed, the loader turns control over to the loaded program.

SECTION 3

APPLICATION PROGRAM CHECKOUT AND EXECUTION

Although it is beyond the scope of this manual to give precise procedures for developing specific types of applications, there are general considerations that depend on the kinds of services required by an application that can be described.

After the systems analysis and design for the application has been done, and the actual hardware configuration has been decided upon (i.e., the memory size, and kinds and numbers of peripheral devices), you are ready to develop your application programs.

Section 2 describes the program development procedure. This section presents flow diagrams indicating the debug and execution procedures.

The following considerations should be kept in mind while reading the flowcharts in this section:

- All disk volumes must be initialized and static file space allocated, using offline utility programs, before files are usable by application and system software.
- All program development has been done offline (without Executive software).
- Application programs, whether they are written in FORTRAN, COBOL, assembly language, or a combination, are normally executed with the online Executive software, and use the task, I/O, and clock management facilities of that environment. (Some simple assembly language application programs can be executed offline, see the Executive and Input/Output manual.)
- Initial checkout of applications programs can be done in an offline environment with the Debugger utility program; see the Utility Programs manual. Checkout in an online environment requires the Online Debug Program; see the Planning and Building an Online Application manual.

Figure 3-1 shows the offline debugging process for an application program. The assumption is made that the individual modules of an application can be debugged initially without the presence of the Executive modules.

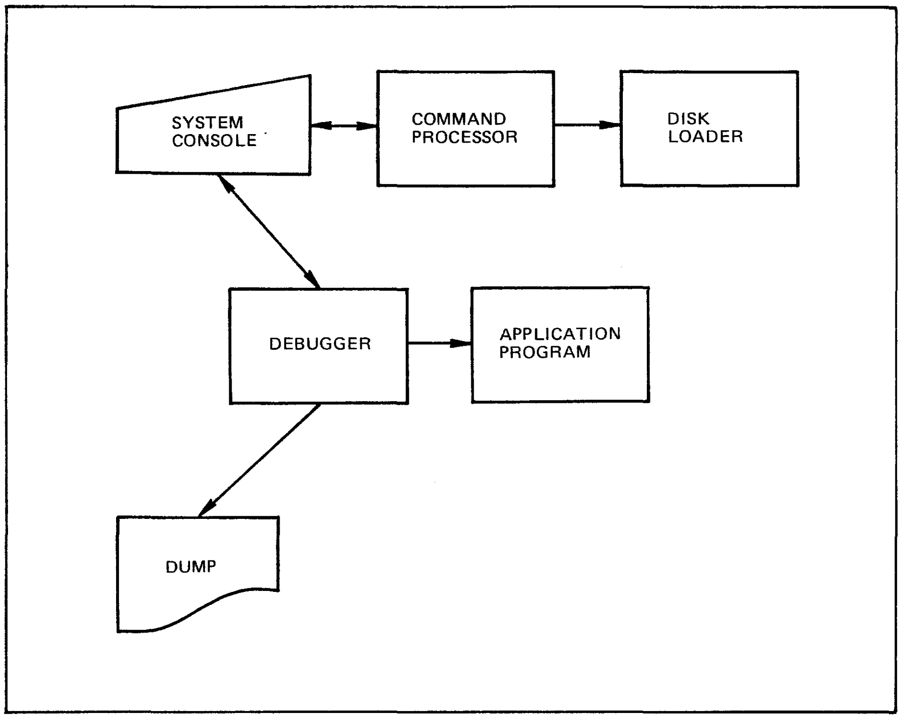


Figure 3-1. Offline Debugging of an Application Program

In order to run an application, you establish a specialized software environment for your application when you configure the Level 6 system. Configuration parameters define these devices, files, executive and I/O routines, and trap handlers that are required to support the application. After the application program is loaded and started, it issues system service calls to invoke most executive and I/O functions; however, some functions are requested implicitly or are controlled by the operator.

Figure 3-2 shows the execution process for an application program that has been developed to run in an online environment using some of the services available from Executive software modules. Configuration Load Manager accepts information specifying system characteristics, Executive modules to be loaded, and the application program to be executed.

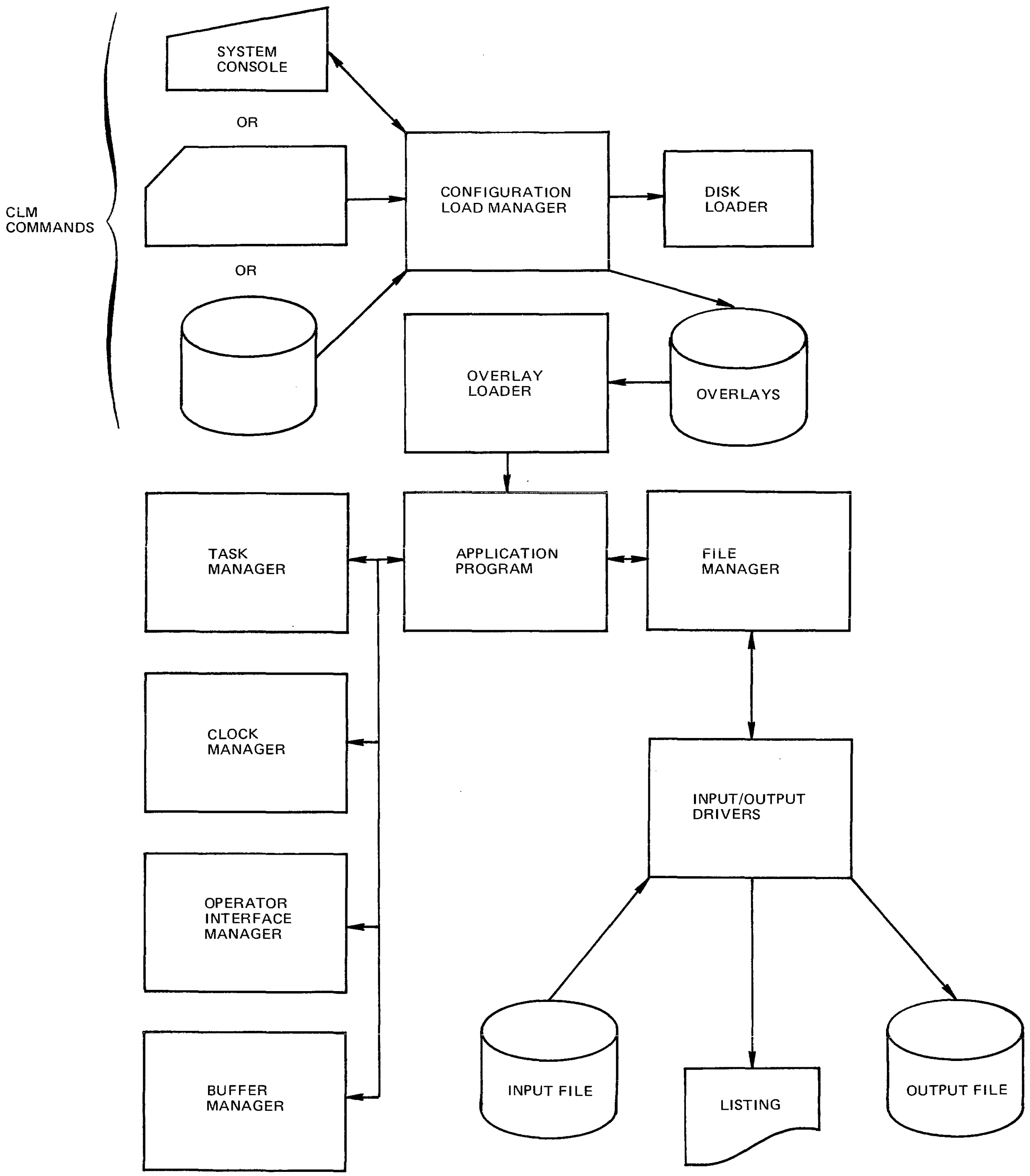


Figure 3-2. Execution of an Online Application Program

Figure 3-3 shows the online debugging process for an application program. The Online Debug Program sets application program breakpoints and trace traps. It transfers debug execution results to a console or KSR-like terminal.

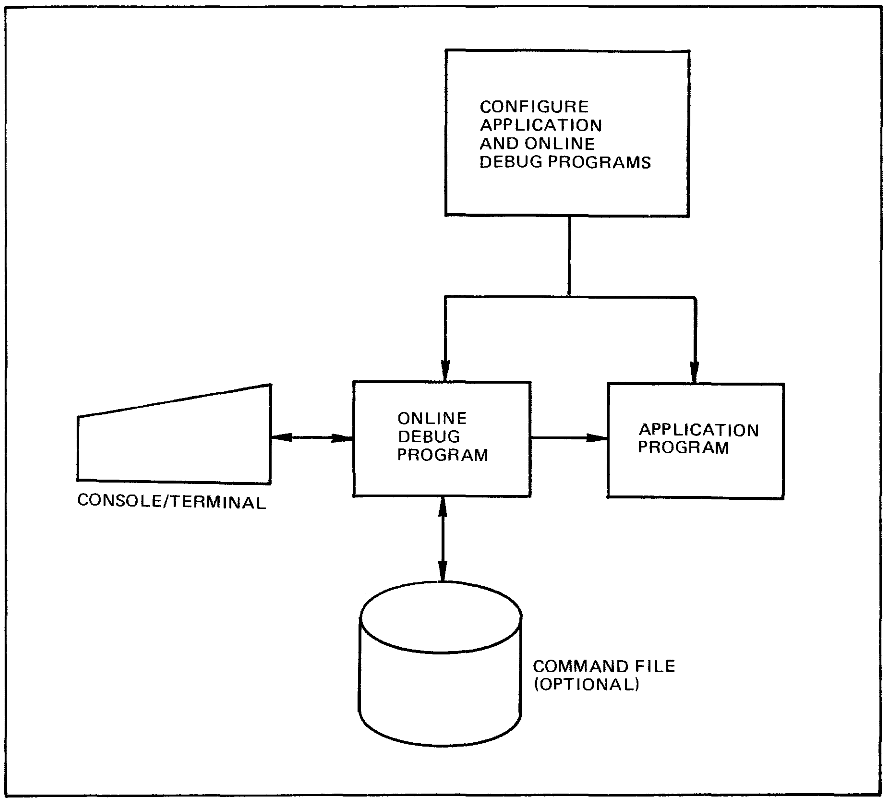


Figure 3-3. Online Checkout of an Application Program

SECTION 4
SYSTEM CONVENTIONS

The standard media formats and system conventions described below provide the basis for an orderly exchange of information between system and application programs, a consistent file system interface for all programs, and a coherent means of developing application programs.

MEDIA CONVENTIONS

Standard data representation formats for data on disk, card, and paper tape are described in the following pages. The term disk includes diskette and cartridge disk devices.

Disk Data Format and Organization

The volume layout, data formats, and file organizations for cartridge disk and diskette are identical. Differences are noted for device-dependent parameter values.

Data is physically organized on cartridge disk into sectors of 256 bytes. There are 24 sectors per track, two tracks per cylinder, and on each volume 204 cylinders for low density packs (100 tracks per inch) or 408 cylinders for high density packs (200 tracks per inch).

Data is physically organized on the diskette into sectors of 128 bytes. There are 26 sectors per track, and 77 tracks per volume.

Contiguous space allocated to a file is referred to as an extent, which consists of an integral number of contiguous sectors. A file can be composed of more than one extent (i.e., physical space for a file need not be contiguous), and logically consists of logical records. A multiextent file, and the relationship between sectors and the logical records of a relative file are illustrated below.

Before data can be written on a new disk, the volume must be prepared to receive it. This initialization of the volume is done by means of an offline utility program that creates a set of standard structures detailing the volume layout and contents.

Prior to program execution, a file could be allocated by using an offline utility. Such a file is static, one that cannot be extended dynamically during execution. A dynamic file, that can be extended during execution, must be created during online execution using File Manager, and must not be allocated offline.

The control structures provided by the volume initialization utility are summarized in Table 4-1.

Table 4-1. Disk Volume Control Structures

Structure	Location		Length (Sectors)	Updated By (Utility Function)
	Track	Sector		
Bootstrap Record	00	00	1	
Intermediate Loader Records	00	01-06	6	
Volume Label	00	07	1	Rename(RN), Initialize (IN)
Volume Index of Defective Sectors	00	08	1	Allocate(AL), Delete(DL), Initialize(IN)
Volume Allocation Bit Map	00	09-10	2	LA,DL,IN
Volume Directory (cartridge disk)	00	11-23	13[a]	IN,AL,DL, Rename(RN), Copy(CP)
Volume Directory (diskette)	00	11-25[a]	15[a]	RN,CP,IN,AL,DL

[a]Default values; they can be modified when the volume is initialized.

The interrelationship of the various control and data structures on a disk is illustrated graphically in Figure 4-1. The location of each structure is enclosed in parentheses, the first value is the track number and the second value the volume-relative sector number. Partitioned files are BES files used by and available only to system programs, whereas applications use relative files. Succeeding paragraphs describe the control structures given in Table 4-1.

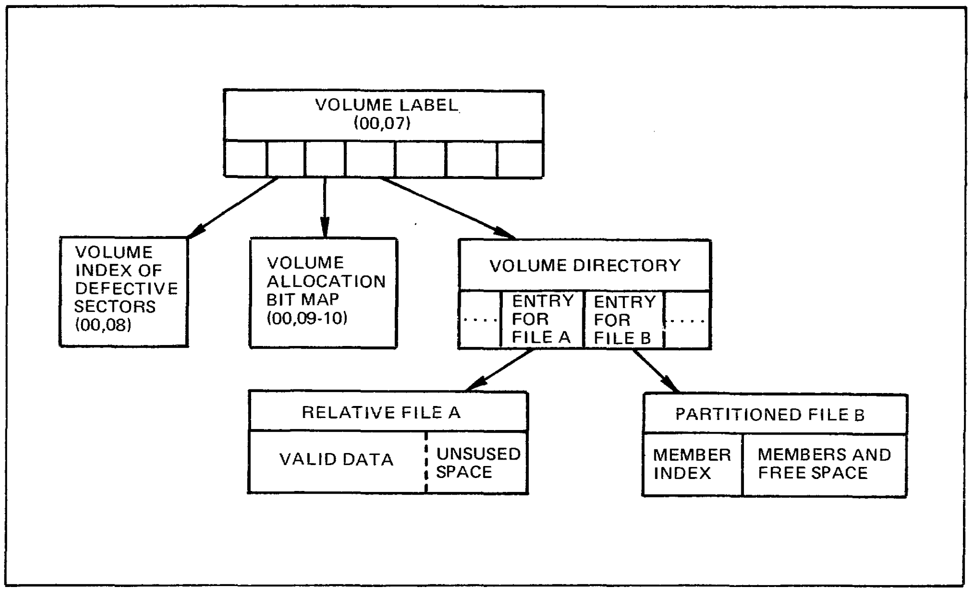


Figure 4-1. Interrelationship of Disk Volume Structures

BOOTSTRAP RECORD

The bootstrap record is created by the Bootstrap Generator in specialized form on sector 0 of track 0 of a disk. The bootstrap record consists of a program that is loaded by a ROM bootstrap loader. This program, in turn, loads intermediate loader records.

INTERMEDIATE LOADER RECORDS

Loader records are placed in sectors 1 through 6 of track 0 of the volume being initialized, and constitute an intermediate loader that loads and specializes the disk loader when needed.

VOLUME LABEL

The volume label occupies sector 7 of track 0; it contains the volume identifier and specific information about the volume and the data recorded on it. The structure is created by Utility Set 1; its contents are shown in Table 4-2. The volume-relative or relative sector number is the position of a sector relative to the beginning of the volume, starting with relative sector number 0.

Table 4-2. Volume Label Contents

Entry	Byte	Number of Bytes	Contents	Explanation
0	0	4	VOL1	Identifies this record as a volume label.
1	4	6	Volume identifier	A 1- to 6-character ASCII name uniquely identifying this volume.
2	10	1	Accessability byte	A blank signifies public access.
3	11	26		Reserved.
4	37	14	Owner identifier	A 14-character ASCII owner identifier. Default value is 14 blanks.
5	51	25		Reserved.
6	76	2	Sector sequence code	Defines physical sequencing of sectors on a track. A blank indicates sequential order.
7	78	1		Reserved.
8	79	1	G	Indicates a standard BES ASCII label.
9	80	2	Cylinders/ Volume	The number of cylinders per volume (hexadecimal): 00CC - Single density cartridge disk 0198 - Double density cartridge disk 004D - Diskette[a]
10	82	1	Cylinders/ Volume (diskette only)	The number of cylinders per diskette volume (hexadecimal): 4D - Diskette 00 - Cartridge disk
11	83	1	Tracks/Cylinder	The number of tracks per cylinder (hexadecimal): 01 - Diskette 02 - Cartridge disk
12	84	1	Sectors/Track	The number of sectors per track (hexadecimal): 1A - Diskette 18 - Cartridge disk
13	85	1	Sectors/bit of allocation map	The number of sectors represented by each bit in the bit allocation map: 01 - Diskette 08 - Cartridge disk
14	86	2	Sector size	The number of bytes in a sector (hexadecimal): 0080 - Diskette 0100 - Cartridge disk
15	88	2		Reserved

Table 4-2 (cont). Volume Label Contents

Entry	Byte	Number of Bytes	Contents	Explanation
16	90	2	0008	The relative sector number location of the defective sector index.
17	92	2	Defective sector index size	The number of bytes in the defective sector index (hexadecimal): 0080 - Diskette 0100 - Cartridge disk
18	94	2	0009	The location of the volume allocation bit map, given by the relative sector number.
19	96	2	Allocation bit map size	The number of bytes in the volume allocation bit map (hexadecimal): 0100 - Diskette 0200 - Cartridge disk
20	98	2	Start of volume directory	The location of the first sector of the volume directory, given by the relative sector number (hexadecimal). This value is set by a parameter in the Initialize utility. Default: 000B
21	100	2	Volume directory size	The number of bytes in the volume directory (hexadecimal). This value is set by a parameter in the initialize utility. Default: 0780 - Diskette 0D00 - Cartridge disk
22	102	26	Zeros	Reserved (diskette)
		154	Zeros	Reserved (cartridge disk)

[a]For diskette volumes initialized by BES 1 utilities, this value is all ASCII blanks.

VOLUME INDEX OF DEFECTIVE SECTORS

This element, defined by entries 16 and 17 of the volume label, is used by Utility Set 1 to record defective sectors, up to 64 for diskette and up to 128 for cartridge disk. Each word in the index contains either zeros, or the relative sector number of the defective sector.

VOLUME ALLOCATION BIT MAP

This element, defined by entries 18 and 19 of the volume label, indicates sector usage on a disk volume. Each bit of this element corresponds to one sector on diskette or a group of eight sectors on cartridge disk. If the value of the bit is 0, the sector or group of sectors is available; otherwise it is in use.

VOLUME DIRECTORY

The volume directory, defined by entries 20 and 21 of the volume label, is a relative file consisting of 32-byte directory records that contain the attributes for each file on a disk, and 32-byte remote-extent records that describe the extents of a multi-extent file. Table 4-3 gives the format of a directory record and Table 4-4 gives that of a remote-extent record. A hexadecimal FFFE in the first word of a record indicates a remote-extent record; a zero or FFFF indicates that the record is available for use either as a directory or remote-extent record. In a directory record the first word is the first two ASCII characters of a file name. The directory can be placed anywhere on a volume, but must be in sequenctial sectors.

Table 4-3. Volume Directory Record Contents

Entry	Byte	Number of Bytes	Contents	Explanation
0	0	12	File name	A 12-character ASCII name that is compared to a name specified in an OPEN statement.
1	12	2	File Status/ Type word	Individual bits of this word describe file status and type. See Figure 4-2 for definition.
2	14	2	Record length	Record length in bytes.
3	16	4	Relative end of data	Total number of bytes in the file containing valid data.
4	20	2	First extent location	Relative sector number of the first extent.
5	22	2	First extent range	Number of sectors in the first extent.
6	24	2	Second extent location	Relative sector number of second extent, dynamic files only. If value is 0, extent is not used.
7	26	2	Second extent range	Number of sectors in the second extent. Used only for dynamic files. If value is 0, extent is not used.
8	28	2	Third extent location or remote-extent record location	If a data file consists of three extents, this is the third extent location. If a data file consists of more than three extents, this value is the relative record number of a remote-extent record of the directory file.
9	30	2	Third extent range	If a file consists of three extents, this is the third extent range. If a file consists of more than three extents, this value is 0.

0	1	2		4	5	6	7	8		10	11	12 15
s	0	m	RFU	r	1	1	1	e	RFU	d	0	FILE TYPE

Bit 0:	s = 0 if file is shareable; s = 1 if nonshareable.
Bit 1:	0 indicates fixed-length records.
Bit 2:	If this file is a directory, m = 1 indicates that this is the major directory for the volume.
Bit 4:	r = 1 if deletable records are permitted in this file; r = 0 if they are not permitted.
Bit 5:	1 indicates that nonsequential access to this file is permitted.
Bit 6:	1 indicates that read access to this file is permitted.
Bit 7:	1 indicates that write access to this file is permitted.
Bit 8:	e = 1 if the file can be extended dynamically; e = 0 if it is static and cannot be extended.
Bit 10:	d = 1 if the file is a directory; otherwise, d = 0
Bit 11:	0 indicates that this is a disk file.
Bits 12-15:	File type values are: 0010 for a relative file, 0011 for a partitioned file, 0000 for a directory, 0101 for a relative file with deletable records.

Figure 4-2. File Status/Type Word

Table 4-4. Remote Extent Record Contents

Entry	Byte	Number of Bytes	Contents	Explanation
0	0	2	FFFE	Remote extent record identification code.
1	2	2		Reserved.
2	4	2	First extent location	Relative sector number of the first extent described in this record.
3	6	2	First extent range	Number of sectors in the first extent.
4	8	2	Second extent location	Relative sector number of the second extent described in this record.
5	10	2	Second extent range	Number of sectors in the second extent.
6	12	2	Third extent location	Relative sector number of the third extent described in this record.
7	14	2	Third extent range	Number of sectors in the third extent.
8	16	2	Fourth extent location	Relative sector number of the fourth extent described in this record.
9	18	2	Fourth extent range	Number of sectors in the fourth extent.
10	20	2	Fifth extent location	Relative sector number of the fifth extent described in this record.
11	22	2	Fifth extent range	Number of sectors in the fifth extent.

Table 4-4 (cont). Remote Extent Record Contents

Entry	Byte	Number of Bytes	Contents	Explanation
12	24	2	Sixth extent location	Relative sector number of the sixth extent described in this record.
13	26	2	Sixth extent range	Number of sectors in the sixth extent.
14	28	2	Seventh extent location or remote-extent record location	Relative sector number of the seventh extent described in this record. If more extents are required for a data file, this entry is the relative record number of the next remote-extent record.
15	30	2	Seventh extent range	Number of sectors in the seventh extent. This value is 0 if entry 14 points to a remote-extent record.

A static file is a single extent file with no entries in its directory record of Table 4-3, beyond entry 5. A dynamic file can consist of more than one extent, and the directory record in Table 4-3 has entries to locate up to three extents of one file. When more than three extent location entries are required, entry 8 contains the relative record number within the directory file of a remote-extent record in which the third and up to seven extent location entries can be entered; another remote-extent record is used, if more extents are required.

The directory for the volume illustrated in Figure 4-3 contains records for three files of which the file B extent entries are shown. Entries for extents 1 and 2 point to assigned disk space; entry 8 contains the relative record number of a remote-extent record. Entries 2 and 4 of the remote-extent record point to extents 3 and 4 of file B, whereas the contents of entry 6, 0000, indicates that there are no more extents.

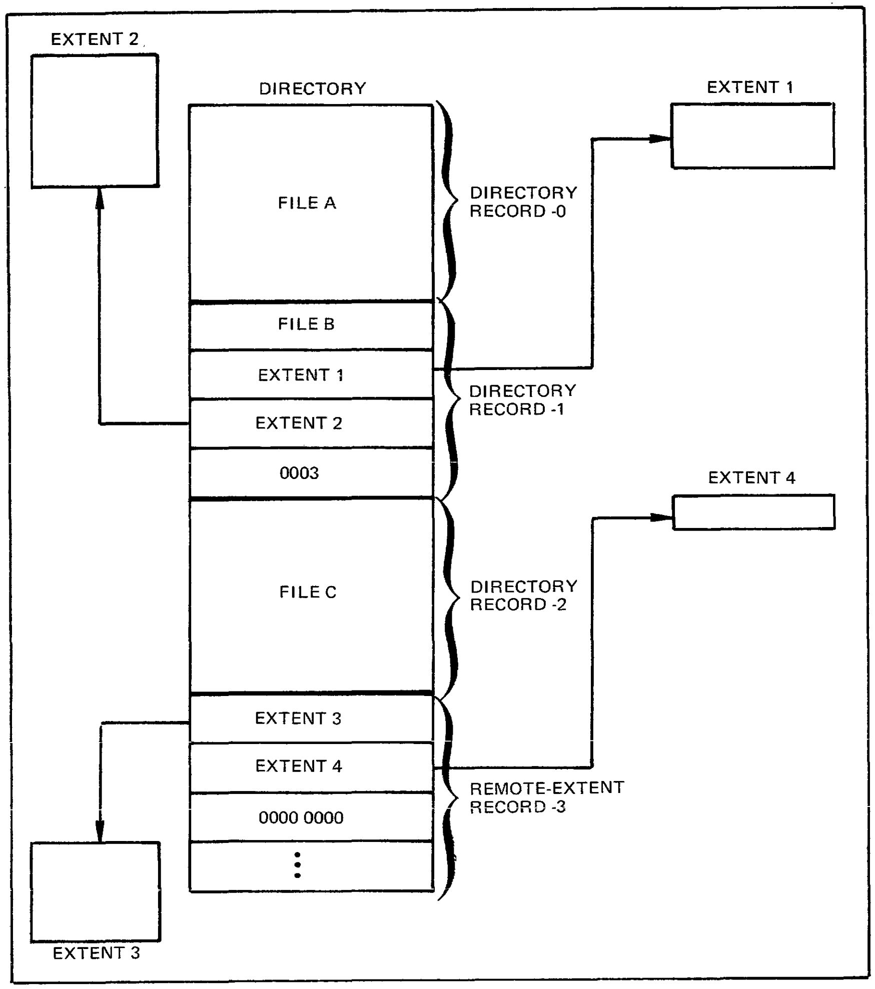


Figure 4-3. Multi-Extent Dynamic File

DISK DATA FILE ORGANIZATION

The organization of application data files on disk supported by File Manager is known as relative file organization. A relative file is logically divided into fixed-length records where each record can be identified by its position relative to the beginning of the file. Within one program, a relative file can be accessed sequentially, by reading or writing the next sequential logical record, or directly by reading or writing any record by supplying a relative record number (i.e., its relative record position in the file).

Given a relative record number and the known fixed record size, File Manager calculates the physical position in the file where the record occurs. However, when using direct access, each program must contain its own technique for determining the relative record number of the desired record.

A program cannot read data beyond the last record position ever written. If it tries, the end-of-data status is returned in the File Manager status word. A program cannot write beyond a static file's "end-of-allocated-space" or beyond the current maximum record value set after open, whichever comes first. However, in a dynamic file it can append (write) additional records.

Figure 4-4 depicts a single extent relative file on diskette containing twelve 80-byte records. The file begins and ends on a sector boundary since a file extent contains an integral number of sectors. Relative records 1, 3, 4, 6, 9, and 11 span sector boundaries; relative record 4 also spans a track boundary. The end of relative record 7 coincides with the end of track 9, sector 01; both are 640 bytes from beginning of the file. Since the end of record 11 does not coincide with the boundary of sector 04, the nondata portion, the last 120 bytes, of the sector is filled with zeros.

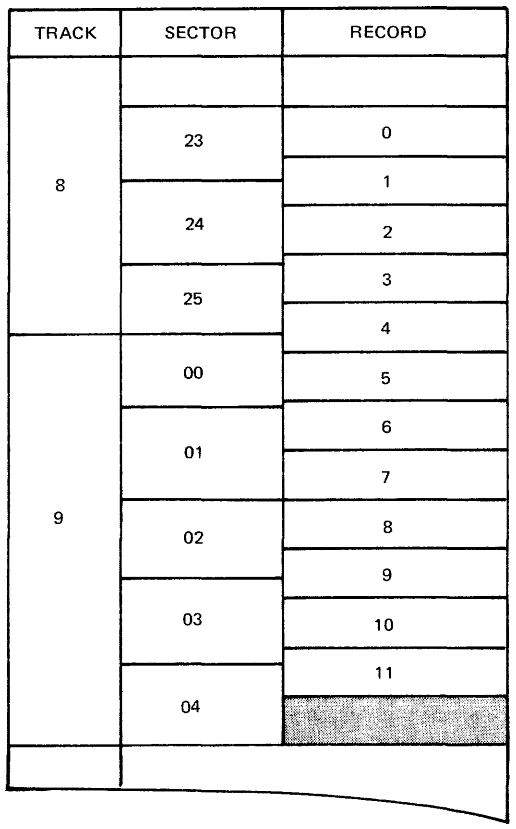


Figure 4-4. Relative File Organization on Diskette (Example)

Using Figure 4-4 as a sample file, a program accesses relative record 6 of the file directly by providing File Manager with the relative record number 6. Record 7 can then be accessed sequentially by requesting the next record or directly by requesting record 7. Either type of access can continue to be used, as suits the application. To access a record out of sequence, direct access must be used.

To provide a program with a requested record, File Manager reads a sector or part of a sector as required, deblocks it (breaks it into its logical records), and places the requested record in the program's buffer. To obtain relative record 4 of the file in Figure 4-4, File Manager reads sector 25 of track 8 and extracts the first 64 bytes of relative record 4, and places them in the program's buffer. Then it reads the first 16 bytes of track 9 sector 00 directly into the end of the program's buffer. By comparison, if physical I/O is used to read the file, the application program must read in the sectors and deblock them.

File Manager supports two types of relative files: one allows and the other does not allow deleted records. A nondeletable-record file consists solely of data. When this file type is created, it is the program's responsibility to populate the file with meaningful data. A program cannot delete records from such a file.

Each record in a deletable-record file contains two nondata bytes to indicate deleted record status. Consequently, if a file's record size is chosen to be one sector, the number of bytes available for data is 126 on diskette and 254 on cartridge disk. When the file type is created, each record position is flagged as deleted. The delete status is changed when a record is written with data. Data of a record whose status is deleted will be read into the program's buffer, and a "deleted record detected" status error is returned. It is the program's responsibility to check for deleted records. (The COBOL I/O routine ignores deleted records on a sequential "read next" request and only reads nondeleted records; if a direct request for a deleted record is made, it returns an invalid key condition.)

Paper Tape Data Format and Organization

Data punched on ASR paper tape is organized into blocks as shown in Figure 4-5 below. Each block consists of:

- A preamble that identifies the beginning of the block
- User-supplied data
- An end-of-block mark

Data residing on paper tape may be read or punched in either binary or ASCII format. The differences between these two formats are also shown in Figure 4-5. For example, the preamble for an ASCII block consists only of NUL characters, whereas the binary format preamble contains the filler character SOH, that separates the NUL characters from the text.

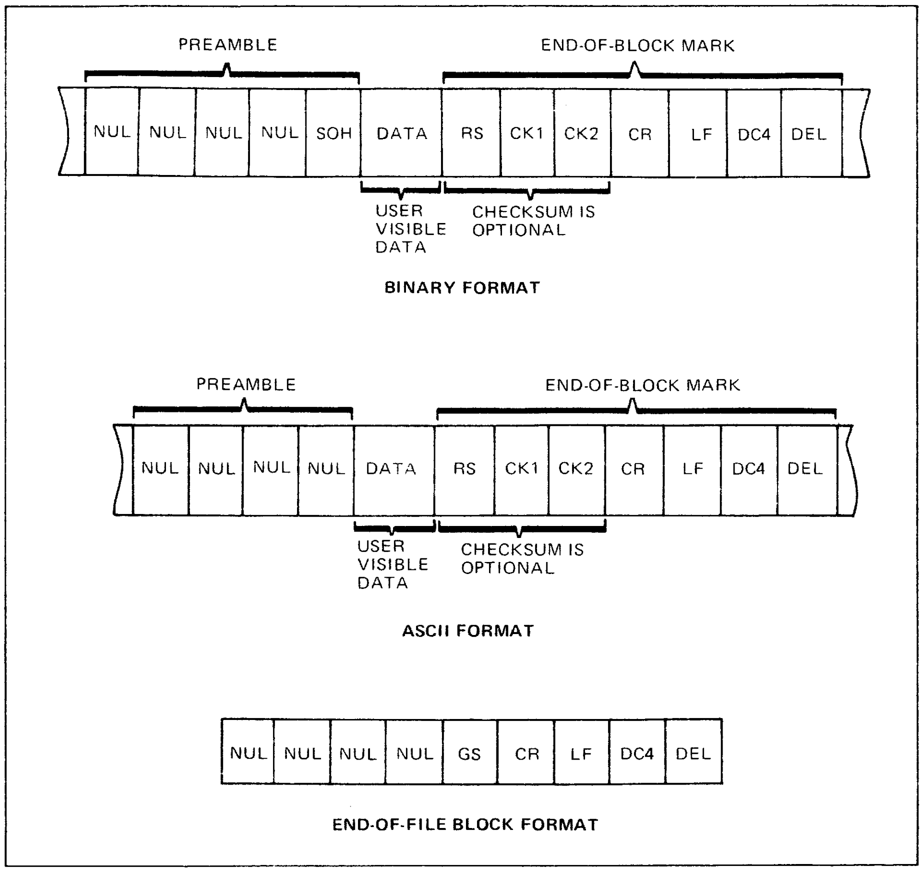


Figure 4-5. Paper Tape Data Organization

The special characters appearing in the preamble and the end-of-file frames are described in Table 4-5. These characters have special control functions and should not be used within the data text portion of a block. If they do appear as part of the text, they must be "escaped," that is, altered, to prevent the triggering of unwanted operations during data transmission.

Note that one end-of-file (EOF) block (Figure 4-6) indicates the end of file; two consecutive EOF blocks indicate end of file and end of recorded data.

Table 4-5. Paper Tape Special Characters

Character	Hexadecimal Equivalent[a]	Escaped Values[a]	Meaning
NUL	00	(30)	Filler used in the preamble
EOT	84	(B4)	ASR control; stops reader when echoed
ENQ or WRU	05	(35)	ASR control; stops reader when echoed
LF	0A	(3A)	Console control: line feed used in the end-of-block frames
CR	8D	(BD)	Console control: carriage return, used in end-of-block frames
DC1 or X-ON	11	(21)	ASR control: turns reader on when echoed
DC2 or TAPE	12	(22)	ASR control: turns punch on when echoed
DC3 or X-OFF	93	(A3)	ASR control: turns reader off when echoed
DC4 or ~~TAPE~~	14	(24)	ASR control: turns punch off when echoed
GS	1D	(2D)	First character of EOF record
RS	1E	(2E)	Separates data characters from checksum characters
@	C0	(F0)	Used to cancel the previously entered character
CAN	18	(28)	Used to cancel line just entered from keyboard
| or ¦	FC	(CC)	Used to indicate that the following characters are "escaped"
DEL	FF	(CF)	Used in the end-of-block frames

[a]Shown with even parity; only low seven bits are significant (e.g., 8D or 0D represent CD) and become, when escaped, BD or 3D respectively.

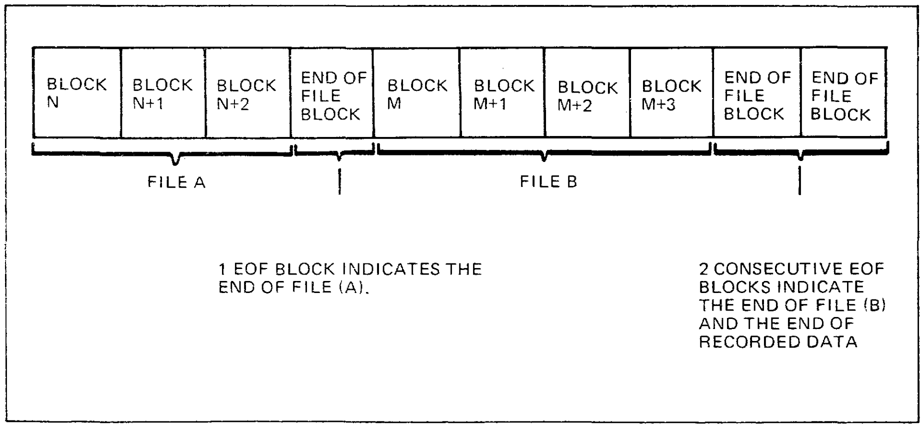

BLOCK N
BLOCK N+1
BLOCK N+2
END OF FILE BLOCK
BLOCK M
BLOCK M+1
BLOCK M+2
BLOCK M+3
END OF FILE BLOCK
END OF FILE BLOCK
FILE A
FILE B
1 EOF BLOCK INDICATES THE END OF FILE (A).
2 CONSECUTIVE EOF BLOCKS INDICATE THE END OF FILE (B) AND THE END OF RECORDED DATA

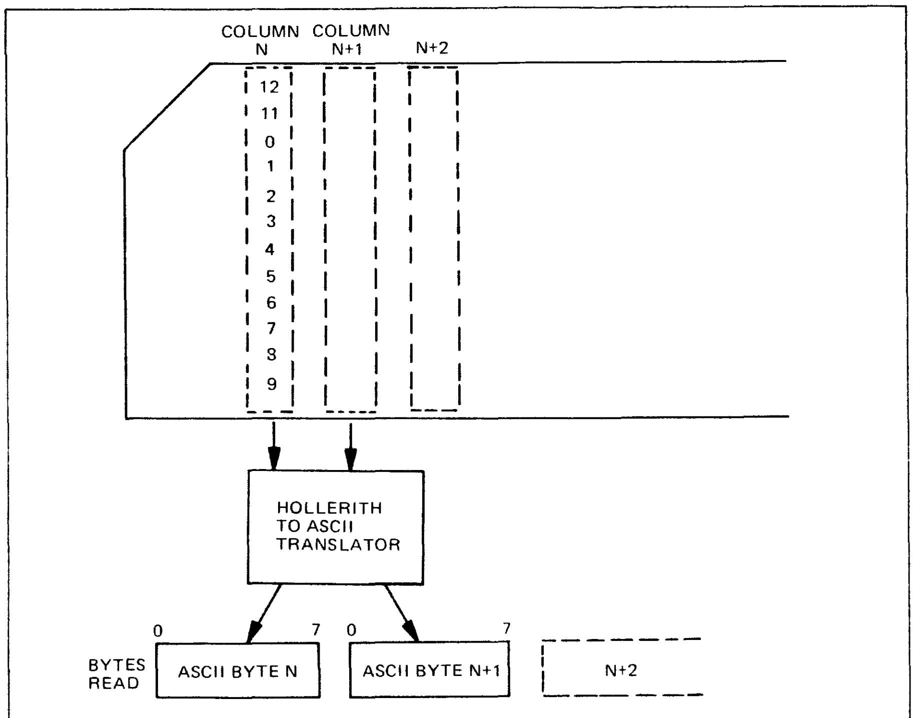
Figure 4-6. Paper Tape File Organization

Note further, that there are two additional special characters that do not require "escaping": SOH, with the hexadecimal value 81, used as a filler character in the preamble, and the backslash (\) with the hexadecimal value of 5C, that indicates the following character to be a binary constant.

Card Data Format and Organization

Data residing on cards may be read in either of two formats: ASCII, or verbatim. These formats are illustrated in Figures 4-7 and 4-8. Table 4-6 shows the Hollerith and ASCII equivalence codes.

The end-of-file indicator for a card deck consists of a card containing the ASCII character "GS" punched in column 1 (the Hollerith equivalent is: 11-9-8-5 punch in column 1); two consecutive EOF cards indicate the end of the card deck as shown in Figure 4-9.



NOTES: 1. This translator will provide a status indicator which will be set whenever an illegal Hollerith code is read.

2. The translator shown above is in the card reader attachment.

Figure 4-7. ASCII Mode Format for Card Data

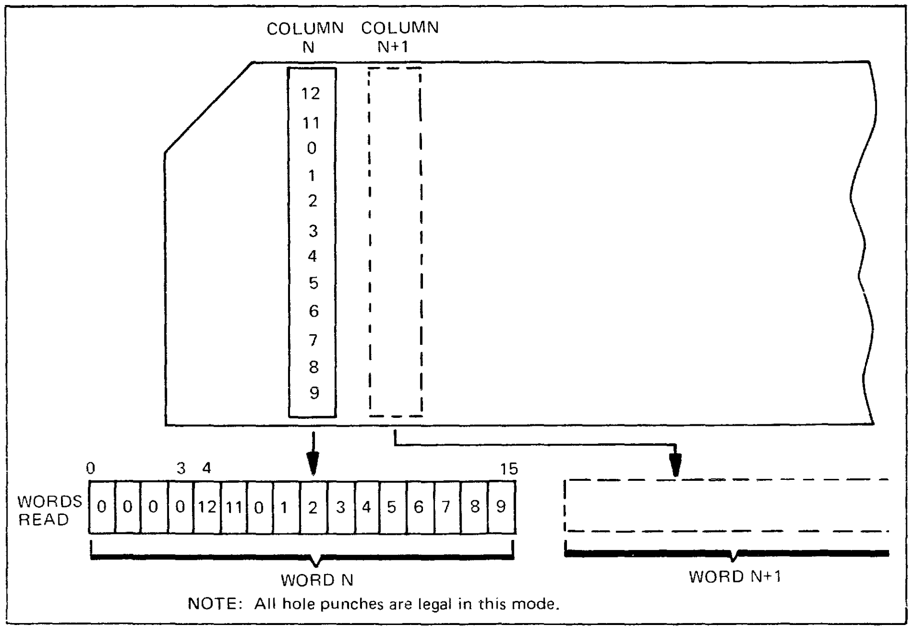


Figure 4-8. Verbatim Mode Format for Card Data

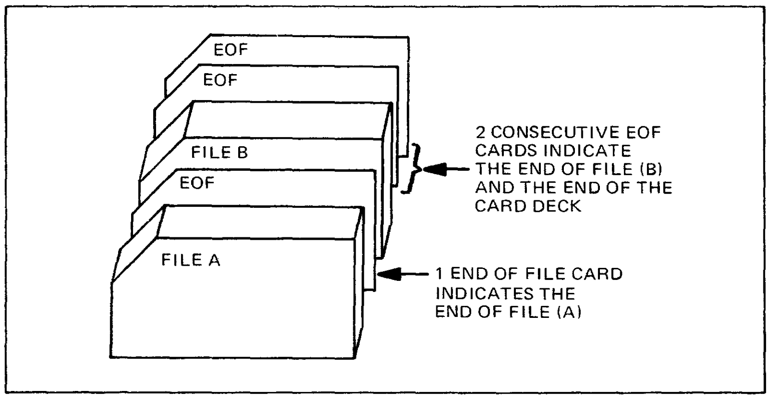

EOF
EOF
FILE B
EOF
FILE A
2 CONSECUTIVE EOF CARDS INDICATE THE END OF FILE (B) AND THE END OF THE CARD DECK
1 END OF FILE CARD INDICATES THE END OF FILE (A)

Figure 4-9. Card File Organization

Character Set and Code Conversion

Table 4-7 shows the standard BES character set, and the ASCII to hexadecimal conversion values.

The standard printable 64-character set consists of:

- Letters A through Z
- Numerals 0 through 9
- Special characters: ! " # $ % & ' () + , - . / : ; < = > ? _ [] \ *
  @ ∧ SP(space)

Table 4-6. Hollerith to ASCII Equivalence Codes

	b8	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	
	b7	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	
	b6	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	
	b5	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	
b4b3b2b1	ROW \ COL	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	COL / ROW
0000	0	NUL 12-0-9-8-1	DLE 12-11-9-8-1	SP NO PCH	0 0	@ 8-4	P 11-7	\` 8-1	p 12-11-7	11-0-9-8-1	12-11-0-9-8-1	12-0-9-1	12-11-9-8	12-11-0-9-6	12-11-8-7	12-11-0-8	12-11-9-8-4	0
0001	1	SOH 12-9-1	DC1 11-9-1	! ① 12-8-7	1 1	A 12-1	Q 11-8	a 12-0-1	q 12-11-8	0-9-1	9-1	12-0-9-2	11-8-1	12-11-0-9-7	11-0-8-1	12-11-0-9	12-11-9-8-5	1
0010	2	STX 12-9-2	DC2 11-9-2	" 8-7	2 2	B 12-2	R 11-9	b 12-0-2	r 12-11-9	0-9-2	11-9-8-2	12-0-9-3	11-0-9-2	12-11-0-9-8	11-0-8-2	12-11-0-8-2	12-11-9-8-6	2
0011	3	ETX 12-9-3	DC3 11-9-3	# 8-3	3 3	C 12-3	S 0-2	c 12-0-3	s 11-0-2	0-9-3	9-3	12-0-9-4	11-0-9-3	12-0-8-1	11-0-8-3	12-11-0-8-3	12-11-9-8-7	3
0100	4	EOT 9-7	DC4 9-8-4	$ 11-8-3	4 4	D 12-4	T 0-3	d 12-0-4	t 11-0-3	0-9-4	9-4	12-0-9-5	11-0-9-4	12-0-8-2	11-0-8-4	12-11-0-8-4	11-0-9-8-2	4
0101	5	ENQ 0-9-8-5	NAK 9-8-5	% 0-8-4	5 5	E 12-5	U 0-4	e 12-0-5	u 11-0-4	11-9-5	9-5	12-0-9-6	11-0-9-5	12-0-8-3	11-0-8-5	12-11-0-8-5	11-0-9-8-3	5
0110	6	ACK 0-9-8-6	SYN 9-2	& 12	6 6	F 12-6	V 0-5	f 12-0-6	v 11-0-5	12-9-6	9-6	12-0-9-7	11-0-9-6	12-0-8-4	11-0-8-6	12-11-0-8-6	11-0-9-8-4	6
0111	7	BEL 0-9-8-7	ETB 0-9-6	' 8-5	7 7	G 12-7	W 0-6	g 12-0-7	w 11-0-6	11-9-7	12-9-8	12-0-9-8	11-0-9-7	12-0-8-5	11-0-8-7	12-11-0-8-7	11-0-9-8-5	7
1000	8	BS 11-9-6	CAN 11-9-8	(12-8-5	8 8	H 12-8	X 0-7	h 12-0-8	x 11-0-7	0-9-8	9-8	12-8-1	11-0-9-8	12-0-8-6	12-11-0-8-1	12-0-9-8-2	11-0-9-8-6	8
1001	9	HT 12-9-5	EM 11-9-8-1	) 11-8-5	9 9	I 12-9	Y 0-8	i 12-0-9	y 11-0-8	0-9-8-1	9-8-1	12-11-9-1	0-8-1	12-0-8-7	12-11-0-1	12-0-9-8-3	11-0-9-8-7	9
1010	10	LF 0-9-5	SUB 9-8-7	* 11-8-4	: 8-2	J 11-1	Z 0-9	j 12-11-1	z 11-0-9	0-9-8-2	9-8-2	12-11-9-2	12-11-0	12-11-8-1	12-11-0-2	12-0-9-8-4	12-11-0-9-8-2	10
1011	11	VT 12-9-8-3	ESC 0-9-7	+ 12-8-6	; 11-8-6	K 11-2	[12-8-2	k 12-11-2	{ 12-0	0-9-8-3	9-8-3	12-11-9-3	12-11-0-9-1	12-11-8-2	12-11-0-3	12-0-9-8-5	12-11-0-9-8-3	11
1100	12	FF 12-9-8-4	FS 11-9-8-4	, 0-8-3	< 12-8-4	L 11-3	\ 0-8-2	l 12-11-3	¦ 12-11	0-9-8-4	12-9-4	12-11-9-4	12-11-0-9-2	12-11-8-3	12-11-0-4	12-0-9-8-6	12-11-0-9-8-4	12
1101	13	CR 12-9-8-5	GS 11-9-8-5	- 11	= 8-6	M 11-4	] 11-8-2	m 12-11-4	} 11-0	12-9-8-1	11-9-4	12-11-9-5	12-11-0-9-3	12-11-8-4	12-11-0-5	12-0-9-8-7	12-11-0-9-8-5	13
1110	14	SO 12-9-8-6	RS 11-9-8-6	. 12-8-3	> 0-8-6	N 11-5	^ ② 11-8-7	n 12-11-5	~ 11-0-1	12-9-8-2	9-8-6	12-11-9-6	12-11-0-9-4	12-11-8-5	12-11-0-6	12-11-9-8-2	12-11-0-9-8-6	14
1111	15	SI 12-9-8-7	US 11-9-8-7	/ 0-1	? 0-8-7	O 11-6	_ 0-8-5	o 12-11-6	DEL 12-9-7	11-9-8-3	11-0-9-1	12-11-9-7	12-11-0-9-5	12-11-8-6	12-11-0-7	12-11-9-8-3	EO 12-11-0-9-8-7	15

① may be " | "

② mav be " ¬ "

③ The top line in each entry to the table represents an assigned character (Columns 0 to 7).
The bottom line in each entry is the corresponding card hole-pattern.

The 96-character set contains the following additional characters (when they are used where only 64 characters are supported, the corresponding upper-case characters are displayed or read):

- Letters a through z
- Special characters: ` { ¦ } ~ DEL[1]

The 128-character set contains the following additional control characters.

NUL[2]	NULL
SOH	Start of heading; filler character for paper tape
STX	Start of text
ETX	End of text
EOT[2]	End of transmission
ENQ[2]	Inquiry
ACK	Acknowledge
BEL	Bell
BS	Backspace
HT	Horizontal tabulation
LF[2]	Line feed
VT	Vertical tabulation
FF	Form feed
CR[2]	Carriage return
SO	Shift-out
SI	Shift-in
DLE	Data link escape
DC1 DC2 DC3 DC4 [2]	Device control characters
NAK	Negative acknowledge
SYN	Synchronous idle
ETB	End of transmission block
CAN[2]	Cancel
EM	End of medium
SUB	Substitute
ESC	Escape
FS	File separator
GS[2]	Group separator
RS[2]	Record separator
US	Unit separator

[1]DEL is a control character used for paper tape.

[2]These characters have particular uses for paper tape.

Table 4-7. Standard Character Set and ASCII to Hexadecimal Conversion

ASCII character = Two 8-bit hexadecimal digits: H1 H2

H2 \ H1	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	¦
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	∧	n	~
F	SI	US	/	?	O	_	o	DEL

PROGRAMMING CONVENTIONS

The following conventions are provided so that you can design your application programs to interface smoothly with Executive software.

The register conventions are particularly important since Executive software uses certain registers without saving the contents. If your programs use the same registers, you must save the information for later use by your own program.

Module and File Name Conventions

System software module names begin with a Z character; to avoid possible duplication, no application program should use Z as the first character in a module or symbol name. System module names are six characters in length; the second character defines the major category to which the module belongs. Table 4-8 lists the categories of system software and the first two characters of the name.

The names of program modules that are processed by program development tools (compiler, assembler, and so on), are given a suffix by the particular component doing the processing. Table 4-9 lists these suffixes.

See the Program Development Tools manual for information about using the module suffixes.

Table 4-8. System Module Name Prefixes

Category	Name Prefix
Executive	ZX
Input/Output	ZI
File Manager	ZY
Configuration Load Manager	ZG
Assembler	ZA
FORTRAN Compiler	ZF
COBOL Compiler	ZC
BASIC Compiler	ZB
Macro Preprocessor	ZP
Linker	ZL
Editor	ZE
Utilities	ZU
Communications	ZQ

Table 4-9. System-Program-Assigned Suffixes

Module Type	Suffix
Assembly language source module	.A
FORTRAN language source module	.F
COBOL language source module	.C
BASIC language source module	.B
List module	.L
Link maps	.M
Object module	.O
Macro routine source module (unexpanded)	.P

Calling Sequences

There are standard calling sequences for invoking system services, and for calling external procedures.

SYSTEM SERVICE REQUESTS

Requests for system services (Executive and Input/Output) are made by assembly language programs when the appropriate "link and jump" statement is used that specifies the entry point of the required service routine. The format of a system service request is:

```
LNJ $B5,<entry
```

where "entry" is the external label of the entry point of the particular service routine required; register B5 is always used to store the calling program's return point.

Programs written in higher level languages may also request system services. When they do, the compiler of the language involved translates the source language statement into a code sequence of the required format.

If the system service used by a program returns information to the program the use of registers both by the application program and by the system modules becomes important. It may be necessary for the application program to save the contents of some of the registers it uses, if the system service uses the same ones. See "Register Conventions and Usage" below.

EXTERNAL PROCEDURES

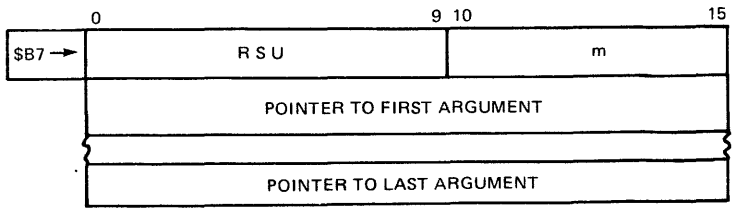
External procedures are those that are assembled or compiled separately from the invoking procedure. These procedures may be either functions, that is, procedures returning a single value to the caller, or subroutines, namely, procedures that alter data contained in an area common to both the procedure and its caller. For example, the FORTRAN mathematial routines (sine, cosine, etc.) are external procedures. You may also write external procedures.

The calling sequence for external procedures (generated by the CALL statement in assembly language and FORTRAN) has the following format:

```
LAB   $B7,list
LNJ   $B5,<entry
```

where "list" represents the label assigned to the argument list, and "entry" is the external label of the subroutine's entry point. The argument list contains the addresses of the parameters to be processed by the called procedure. A standard assembly language or FORTRAN CALL statement causes the argument list to be placed "inline" (immediately after the LNJ instruction); however, a user-written assembly language program could call external procedures using out-of-line argument lists.

An external procedure should always 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 4-10.



R S U: Reserved for system use (must not be modified by called procedure)
m: Number of words in the argument list (m = 1 if there are no argument pointers)

Figure 4-10. Argument List

Register Conventions and Usage

There are 18 visible registers provided for the use of system and user programs. They are:

- R1 through R7 - general registers
- B1 through B7 - address registers
- P - the program counter
- S - the status register
- I - the indicator register
- M1 - the mode control register

All registers are 16 bits in length. (See the Assembly Language manual for a detailed description of these registers.)

The Executive and Input/Output modules generally use the following registers <u>without</u> preserving their contents: I, R1, R2, R6, R7, B3, B4, and B5. If any of the information in these registers is of value to the application program, then the program should save the register contents before making an Executive or Input/Output service request.

The contents of the remaining registers (S, R3, R4, R5, B1, B2, B6, B7, and M1) are not altered by any of the system services.

Table 4-10 summarizes the usage of registers by Executive and Input/Output modules.

Table 4-10. Register Usage by Executive and Input/Output Modules

Register	Contents	Using Component(s)
R1	Return status	Drivers (card reader, printer etc.) Task Manager
	Function code	File Manager
	Error information	File Manager, other system services
	Arguments	System services
R2	Logical File Number	File Manager
	Arguments	System services
	System console channel number	Error reporting routines
R3	Error information	FORTRAN run-time routines
R6	Software status word	Error reporting routines
	Record length, number of bytes transferred, number of records to be spaced or skipped	File Manager

Table 4-10 (cont). Register Usage by Executive and Input/Output Modules

Register	Contents	Using Component(s)
R7	File status	File Manager
	Channel number of a device associated with an I/O error	Error reporting routines
B3	Arguments	System services
	Address of value to be used as a record number	File Manager
B4	IORB address	System services
	Address of the path name of a file	File Manager
	Address of a buffer	Buffer Manager
B5	Return address	System services, external subroutines
B6	Address of the FORTRAN workspace	FORTRAN Run-Time routines
B7	Argument list address	FORTRAN Run-Time routines (also user-written external procedures)
P	Address of the next instruction	Executing programs
S	System status information	Task Manager, other system services
I	Program status indicators	System services
M1	Trap enable control bits	Trap handling routines

Data Types

The following data types are those recognized by the BES software. Refer to the Assembly Language, COBOL, BASIC, or FORTRAN manuals for the specific means of defining the various data types within each language.

Table 4-11 lists the data types available for each language, and relates them to the formats required by Level 6 hardware as described in the Assembly Language manual.

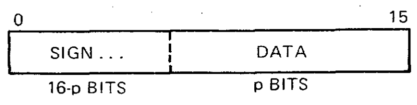
Table 4-11. Summary of Data Types

Data Type	Language Usage				Hardware Format
	Assembly	FORTRAN	BASIC	COBOL	
Short Fixed-point Binary	X	X (Integer)	-	-	Signed integer word
Long Fixed-point Binary	-	X (Integer)	-	-	Signed integer double-word
Short Floating point Binary	X	X (Real)	X	-	Signed integer double-word
Bit String	X (Hex, or bit)	X (Logical)	-	-	Unsigned word, double-word
Character String	X (ASCII)	X (Character, nonvarying)	X	X (Character, nonvarying)	Unsigned byte, word, double-word, etc.
Array	-	X	-	X	All of the above
Unpacked decimal	-	-	-	X	Byte strings
Structure	-	-	-	X	Ordered set of all of above

REAL ARITHMETIC BINARY DATA TYPES

Short Fixed-Point Binary

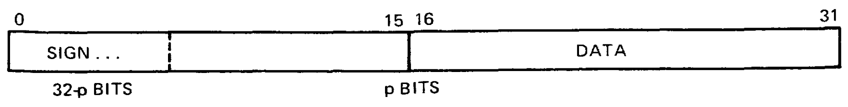
Data of this type is stored aligned on a word boundary. Its precision, p, is: 0<p< 16, and it is represented as a two's complement binary integer stored in a 16-bit word of the form:



SIGN:
ZEROS= PLUS
ONES= MINUS

Long Fixed-Point Binary

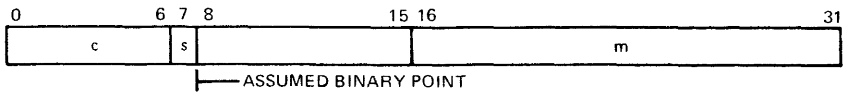
Data of this type is stored aligned on a word boundary. Its precision, p, is: 15 <p< 32, and it is represented as a two's complement binary integer stored in a pair of 16-bit words of the form:



SIGN:
ZEROS= PLUS
ONES= MINUS

Short Floating-Point Binary

Data of this type is stored aligned on a word boundary. Its precision, p, is: $0<p< 24$, and it is represented as a binary fraction m, a sign bit s, and an excess 64 binary integer characteristic c, stored in a pair of 16-bit words of the form:



The characteristic c is not stored in memory as a negative number; if the first bit is a 0, the other bits are stored as the complement of the true value; if the first bit is 1, the characteristic is stored as a binary integer.

The value zero is represented when m = 0, s = 0, and c = 0. For all other values, m satisfies the relationship: $1/16 \leq m< 1$. The value represented by this data type is: $-1^s \times m \times 16^{(c-64)}$.

STRING DATA TYPES

Bit String

Data of this type is stored on a word boundary and occupies n consecutive bits of an integral number of words. The bit positions of the string are numbered from left to right such that bit position 0 is the leftmost bit, and bit position n-1 is the rightmost bit.

Character String

Data of this type may be stored aligned on a word boundary, occupying an integral number of words, or it may be stored unaligned and occupying n consecutive bytes. Each byte contains a single 7-bit ASCII character, right-justified within the byte. The unused bit of each byte must be zero.

ARRAYS

An array is an n-dimensional, ordered collection of scalar data, each element of which has identical attributes.

ERROR REPORTING CONVENTIONS

Error reporting conventions vary depending upon whether or not a particular system has a console available. For those systems using a console, error messages are normally printed out on the console printer; error information is contained in registers in those systems operating without a console.

Components issue four different types of error messages:

- Physical I/O error
- Logical I/O errors
- Component-specific errors
- Informational messages

Excluding informational messages, all other messages are preceded by either a 4- or a 6-digit hexadecimal code. Some components such as the CLM and the loaders issue 4-digit codes regardless of whether the message is printed out or is contained in a register. In general, printed messages are preceded by 6-digit codes. The general format of this code is xxyyzz, where:

xx - Is the component number
yy - Is the error category
zz - Is the specific error code

Table 4-12 summarizes the error codes issued by systems running with and without a console. The specific error codes are tabulated in the Operator's Guide.

Table 4-12. Summary of Error Reporting Conventions

Operating Situation	Printout	Control Panel (Register R1)
System Programs	xxyyzz xx = 10 (Assembler) 11 (Linker) 12 (Utility Sets) 14 (FORTRAN Compiler) 17 (Command Processor) 18 (Cross-Reference Program) 19 (Editor) 21 (Program Patch) 23 (Macro Preprocessor) 25 (Dump Edit) 26 (COBOL Compiler) 30 (Bootstrap Generator) yy = 00 (Logical I/O) 01 (Physical I/O) 0F (Component-specific)	16zz (loader)
Online-with console	13zz (CLM) xxyyzz xx = 15 (FORTRAN Run-Time routines) 20 (Operator Interface Manager) 22 (File Manager) 27 (COBOL Run-Time Routines) 80-FF (User-defined) yy = 01 (Physical I/O) 02 (Logical I/O)	16zz (loaders)

Table 4-12 (cont). Summary of Error Reporting Conventions

Operating Situation	Printout	Control Panel (Register R1)
Online-without console		16zz (loader) 13zz (CLM) 20zz (all other messages)
NOTE: During online operation without a console, if register R1 contains 20zz, the error code xxyyzz can be found by taking the value (address of a buffer area) in register B4 and examining the location pointed to by that register. See the Operator's Guide for the procedure for examining selected areas of memory.		

APPENDIX A
ORGANIZATION OF DISTRIBUTION MEDIA

The following tables provide information about the system software as it distributed on diskette and cartridge disk.

Table A-1. System Software Modules on Diskette

Diskette VOLID	Software Module	Module Name	Approximate Size in Words
S20001	File Name: PROGFILE (Load Modules)		
	Assembler	ASM	16K[a,b]
	Command Processor	CMDPRC	8K[b]
	Cross-Reference Program	XREF	16K[a,b]
	Bootstrap Generator	BTGEN	16K
	Offline Debugger	DEBUG	2000
	Dump Edit	DPEDIT	8K[a,b]
	Macro Preprocessor	MACRO	16K[b]
	Editor	EDIT	16K[a,b]
	Linker	LINKER	16K[a,b]
	Disk Loader	DSKLDR	800
	Offline Trap Handler	TRPHND	150
	Program Patch	PATCH	16K[b]
	Utility Set 1	UTILL1	16K[b]
	Utility Set 2	UTILL2	16K[b]
	Utility Set 3	UTILL3	16K[b]
S20002	File Name: PROGFILE (Load Modules)		
	Command Processor	CMDPRC	8K[b]
	Disk Loader	DSKLDR	800
	Offline Trap Handler	TRPHND	150
	Communications CLM Data Structure Cleanup	ZGQCDS	60
	Buffer Manager	ZXBM01	100
	Executive	ZXEX03	2300
	File Manager	ZYFM02	3200
	Map Utility	ZXMAP	200

Table A-1 (cont). System Software Modules On Diskette

Diskette VOLID	Software Module	Module Name	Approximate Size in Words
S20002 (cont)	File Name: PROGFILE (Load Modules)		
	Online ASR Driver	ZIASR	1200
	Online Card Reader Driver	ZICDR	125
	Cartridge Disk Driver	ZICDSK	225
	Online Diskette Driver	ZIDSK	175
	Online KSR Driver	ZIKSR	450
	Online Printer Driver	ZILPT	125
	Configuration Load Manager	CLM	8K[b]
	CLM Overlays	CLM2 CLMFIL CLMST2 CLMOLK CLMBUF CLMST1 CLMOLA C$CLMBUF C$CLMST1 C$CLMFIL D$CLMFIL D$CLMST2 D$CLMBUF D$CLMST1	
	Online Debugger (Overlay version)	ZDBG	1100
	Debug Overlays	ZDBG00 ZDBG01 ZDBG02 ZDBG03 ZDBG04 ZDBG05 ZDBG06 ZDBG07 ZDBG08 ZDBG09 ZDBG10 ZDBG11 ZDBG12 ZDBG13 ZDBG14 ZDBG15 ZDBG16 ZDBG17	
	Online Debugger (Resident version)	ZDBG1	3000
	File Name: OBJFILE (Object Modules)		
	Online ASR Driver	ZIADON.O	
	Online Card Reader Driver	ZICDON.O	
	Operator Interface Manager (for console)	ZIOIM.O	
	Operator Interface Manager (for control panel)	ZIOIMP.O	
	Online Diskette Driver	ZIDKON.O	
	Online KSR Driver	ZIKDON.O	

Table A-1 (cont). System Software Modules On Diskette

Diskette VOLID	Software Module	Module Name	Approximate Size in Words
S20002 (cont)	File Name: OBJFILE (Object Modules)		
	Online Cartridge Disk Driver	ZIDCON.O	
	Common Subroutines for Online Drivers	ZIOSUB.O	
	Online Printer Driver	ZIPRON.O	
	Buffer Manager Initialization	ZXBIN.O	
	Buffer Manager	ZXBMR.O	
	Clock Manager (basic)	ZXCMGR.O	
	Clock Manager Time and Date Routine	ZXCTDA.O	
	Executive Initialization	ZXIN03.O ZXSEM.O	
	Trace Trap Handler	ZXTRCM.O	
	Task Manager	ZXTSKM.O	
	Error Handler	ZUERR.O	
	Overlay Loader	ZXOVLY.O	
	Offline Diskette Driver	ZIFDD.O	
	Offline KSR Driver	ZIKDFF.O	
	Offline ASR Driver	ZIADFF.O	
	Offline Card Reader Driver	ZICDRF.O	
	Offline Printer Driver	ZIPRTF.O	
	Offline Debugger	DEBUG.O	
	Offline Cartridge Disk Driver	ZICDD.O	
	Map Utility	ZXMAP.O	
	File Name: CLMMAP (Link Maps)		
	Link map of Executive	ZXEX03.M	
	Linker Command File for Executive	ZXEX03.S	
	Link map of Cartridge Disk Driver	ZICDSK.M	
	Link map of Online ASR Driver	ZIASR.M	
	Link map of Online Card Reader Driver	ZICDR.M	
	Link map of Online Debug Program (resident version)	ZDBG1.M	
	Linker Command File for Online Debug Program (resident version)	ZDBG1.S	
	Link map of Online Debug Program (overlay version)	ZDBG.M	
	Linker Command File for Online Debug Program (overlay version)	ZDBG.S	

Table A-1 (cont). System Software Modules on Diskette

Diskette VOLID	Software Module	Module Name	Approximate Size of Words
S20002 (cont)	File Name: CLMMAP (Link Maps)		
	Link maps of Online Debug Program Overlays	ZDBG00.M ZDBG01.M ZDBG02.M ZDBG03.M ZDBG04.M ZDBG05.M ZDBG06.M ZDBG07.M ZDBG08.M ZDBG09.M ZDBG10.M ZDBG12.M ZDBG13.M ZDBG14.M ZDBG15.M ZDBG16.M ZDBG17.M	
	Link map of Online Diskette Driver	ZIDSK.M	
	Link map of Online Printer Driver	ZILPT.M	
	Link map of Online KSR Driver	ZIKSR.M	
	Link map of Buffer Manager	ZXBM01.M	
	Linker Command File for Buffer Manager	ZXBM01.S	
	Link map of Map Utility	ZXMAP.M	
	Link map of File Manager	ZYFM02.M	
	Linker Command File for File Manager	ZYFM02.S	
S20003	File Name: PROGFILE (Load Modules)		
	Command Processor	CMDPRC	8K[b]
	Offline Debugger	DEBUG	2000
	Disk Loader	DSKLDR	800
	FORTRAN Compiler	FORTRAN	16K[a,b]
	Floating-Point Simulator	ZFPSIM	400
	Scientific Branch Simulator	ZFBSIM	250
	Offline Trap Handler	TRPHND	150
	File Name: OBJFILE (Object Modules)		
	Absolute value	ZFABS.O	
	Arccosine	ZFACOS.O	
	Truncation	ZFAINT.O	
	Natural logarithm	ZFALOG.O	
	Common logarithm	ZFAL10.O	
	Choosing largest value	ZFAMA0.O	
	Choosing largest value	ZFAMA1.O	
	Choosing smallest value	ZFAMI0.O	

Table A-1 (cont). System Software Modules on Diskette

Diskette VOLID	Software Module	Module Name	Approximate Size of Words
S20003 (cont)	File Name: OBJFILE (Object Modules)		
	Choosing smallest value	ZFAMI1.O	
	Remaindering	ZFAMOD.O	
	Nearest whole number	ZFANIN.O	
	Arcsine	ZFASIN.O	
	Arctangent	ZFATAN.O	
	Arctangent	ZFATA2.O	
	Cosine	ZFCOS.O	
	Hyperbolic cosine	ZFCOSH.O	
	Positive difference	ZFDIM.O	
	Exponential	ZFEXP.O	
	Type conversion	ZFFLOA.O	
	Absolute value	ZFIABS.O	
	Positive difference	ZFIDIM.O	
	Type conversion	ZFIFIX.O	
	Type conversion	ZFINT.O	
	Transfer of sign	ZFISIG.O	
	Length (character)	ZFLEN.O	
	Choosing largest value	ZFMAX0.O	
	Choosing largest value	ZFMAX1.O	
	Choosing smallest value	ZFMIN0.O	
	Choosing smallest value	ZFMIN1.O	
	Remaindering	ZFMOD.O	
	Nearest integer	ZFNINT.O	
	Transfer of sign	ZFSIGN.O	
	Sine	ZFSIN.O	
	Hyperbolic sine	ZFSINH.O	
	Square root	ZFSQRT.O	
	(Used by SQRT and EXP only)	ZFSTWO.O	
	Tangent	ZFTAN.O	
	Hyperbolic tangent	ZFTANH.O	
	Character move and compare	ZFCFIO.O	
	Computed GO TO	ZFGFIO.O	
	Direct access I/O	ZFDFIO.O	
	Formatted I/O	ZFEFIO.O	
	Integer to Integer exponentiation	ZFXFIO.O	
	Real to Integer exponentiation	ZFYFIO.O	
	Real to Real exponentiation	ZFZFIO.O	

Table A-1 (cont). System Software Modules on Diskette

Diskette VOLID	Software Module	Module Name	Approximate Size of Words
S20003 (cont)	File Name: OBJFILE (Object Modules)		
	Internal (storage) file I/O	ZFIFIO.O	
	Object Time Error routine	ZFIOTE.O	
	Device manipulation I/O	ZFMFIO.O	
	Unformatted I/O	ZFNFIO.O	
	Pause	ZFPFIO.O	
	Sequential Access I/O	ZFSFIO.O	
	General I/O	ZFUFIO.O	
	Return buffer space	DQUEUE.O	
	Product	ZFIAND.O	
	Exclusive OR	ZFIEOR.O	
	Set up space for Control Blocks	ZFINBD.O	
	Inclusive OR	ZFIOR.O	
	Shift	ZFISHF.O	
	Complement	ZFNOT.O	
	Return buffer space	ZFSPWN.O	
	Initiate task after time interval	START.O	
	Acquire permanent buffer space	ZFXBDP.O	
	Obtain date	DATE.O	
	Obtain time of day	TIME.O	
	Initiate task at specified time of day	TRNON.O	
	Suspend task	WAIT.O	
	File status	ZFSTAT.O	
	Bit set	ZFIBSE.O	
	Bit clear	ZFIBCL.O	
	Bit test	ZFIBTE.O	
	Acquire temporary buffer space	ZFXBDT.O	
S20014	File Name: CLMCOMM (Load Modules)		
	MLCP Loader	ZQMLIN	
	Communications Supervisor	ZQEXEC	
	MLP Driver	ZQMLON	
	CLM Action Routines	COMM	
	CLM Clean up	C$COMM	
	CLM Directory	D$COMM	
	TTY Line Protocol Handler	ZQPTTY	
	VIP Line Protocol Handler	ZQPVIP	
	BSC Line Protocol Handler	ZQPBSC	

Table A-1 (cont). System Software Modules on Diskette

Diskette VOLID	Software Module	Module Name	Approximate Size of Words
S20014 (cont)	File Name: MACLIB (Communications Macro Source Modules)		
	Macro Prototype	A14	
	Code Generation macro	BYTE	
	MLCP Op-code macro	B	
	MLCP Op-code macro	BET	
	MLCP Op-code macro	BZT	
	MLCP Op-code macro	BLCT	
	MLCP Op-code macro	BLBT	
	MLCP Op-code macro	BART	
	MLCP Op-code macro	BS	
	MLCP Op-code macro	BEF	
	MLCP Op-code macro	BZF	
	MLCP Op-code macro	BLCF	
	MLCP Op-code macro	BLBF	
	MLCP Op-code macro	BARF	
	MLCP Op-code macro	GNB	
	MLCP Op-code macro	IN	
	MLCP Op-code macro	OUT	
	MLCP Op-code macro	SEND	
	MLCP Op-code macro	RECV	
	MLCP Op-code macro	NOP	
	MLCP Op-code macro	WAIT	
	MLCP Op-code macro	SFS	
	MCLP Op-code macro	CCH	
	MLCP Op-code macro	DEC	
	MLCP Op-code macro	RET	
	MLCP Op-code macro	SR	
	MLCP Op-code macro	SI	
	MLCP Op-code macro	LD	
	MLCP Op-code macro	ST	
	MLCP Op-code macro	C	
	MLCP Op-code macro	AND	
	MLCP Op-code macro	OR	
	MLCP Op-code macro	XOR	
	MLCP Op-code macro	TLU	
	MLCP Op-code macro	LOC	
	MLCP Op-code macro	ORG	
	MLCP Op-code macro	MORG	
	MLCP Op-code macro	DATA	

Table A-1 (cont). System Software Modules on Diskette

Diskette VOLID	Software Module	Module Name	Approximate Size of Words
S20016	File Name: PROGFILE (Load Modules)		
	Disk Loader	DSKLDR	800
	Command Processor	CMDPRC	8K[b]
	Offline Debugger	DEBUG	2000
	Trap Handler	TRPHND	150
	Compiler driver and I/O module	COBOL	16K[a,b]
	Lexical phase	ZCLEX	
	Data Division syntax phase	ZCDD	
	Identification/Environment Division syntax phase	ZCID	
	Allocation phase	ZCALOC	
	Data map listing phase	ZCMAP	
	Replacement phase	ZCRS	
	Diagnostic listing phase	ZCDIAG	
	Procedure Division syntax phase	ZCPD	
	Code generator phase	ZCGEN	
	File Name: ZCRT (object Modules)		
	All I/O statements	ZCRTIO.O	
	ADD statement	ZCRTAU.O	
	Class alphabetic condition	ZCRTBU.O	
	Numeric comparisons	ZCRTCU.O	
	DIVIDE statement	ZCRTDU.O	
	Move edited	ZCRTEU.O	
	Convert decimal to binary	ZCRTFU.O	
	Sign handling	ZCRTGU.O	
	Alphanumeric comparisons	ZCRTLU.O	
	MULTIPLY statement	ZCRTMU.O	
	CLASS NUMERIC condition	ZCRTNU.O	
	SUBTRACT statement	ZCRTSU.O	
	Move numeric	ZCRTVU.O	
	ACCEPT statement	ZCRTPU.O	
	DISPLAY statement	ZCRTYU.O	
	STOP RUN statement	ZCSTOP.O	
	Run time errors	ZCRTER.O	
S20017	File Name: PROGFILE (Load Modules)		
	Disk loader	DSKLDR	800
	BASIC interpreter	BASIC	16K[b,c]
	Command processor	CMDPRC	

Table A-1 (cont). System Software Modules on Diskette

Diskette VOLID	Software Module	Module Name	Approximate Size of Words
S20017 (cont)	File Name: PROGFILE (Load Modules)		
	Offline trap handler	TRPHND	150
	Offline debugger	DEBUG	2000

[a]Automatically uses memory up to 64K

[b]Minimum memory required

[c]Automatically uses memory up to 32K

Table A-2. System Software in Source Module Format on Diskette

Diskette VOLID	Contents
S20004	Assembler
S20005	Offline Drivers, I/O Service for system programs
S20006	Editor, Cross-Reference Program, Offline Debugger, Dump Edit
S20007	Diskette Loader, Offline Trap Handler, Command Processor, Program Patch, Bootstrap Generator
S20008	Utility Set 1, Utility Set 2, Utility Set 3 subroutines
S20009	Configuration Load manager
S20010	Task Manager, Clock Manager, Operator Interface Manager, Trace Trap Handler, Error Handler, Buffer Manager, Online Driver Subroutines, Executive Initialization Routines, Online Drivers, Map Utility, Overlay Loader
S20011	File Manager
S20012	FORTRAN Compiler
S20013	FORTRAN Run-time I/O Routines, FORTRAN Functions, Floating-Point Simulator, Scientific Branch Simulator
S20015	Macro Preprocessor
S20018	Online Debug Program
S20019	Utility Set 1, Utility Set 2, Utility Set 3 Main Programs, Linker
S20020	Communications
S20021	Communications
S20022	Communications

Table A-3. System Software on Cartridge Disk

Cartridge Disk VOLID	Contents
S20051	Executable modules ordered by customer (200TPI cartridge disk)
S20052	Source modules ordered by customer (200TPI cartridge disk)
S20053	Executable modules ordered by customer (100TPI cartridge disk)
S20054 and S20055	Source modules ordered by customer (100TPI cartridge disk)

APPENDIX B
EQUIPMENT REQUIREMENTS

MINIMUM EQUIPMENT FOR PROGRAM DEVELOPMENT

The following equipment is required for program development:

- 6/34 or 6/36 Central Processor with a full control panel and at least 16K words of memory.
- Dual diskette (DIU9102) or cartridge disk (CDU9101, CDU9102, CDU9103, or CDU9104) attached through a Multiple Device Controller or Mass Storage Controller, respectively.
- Console device (TTU9101, TTU9102, DKU9101, DKU9102, or TWU9101) attached through a Multiple Device Controller.
- A printer is desirable but not required (PRU9101, PRU9102, PRU9103, PRU9104, PRU9105, or PRU9106).

MINIMUM EQUIPMENT FOR ONLINE APPLICATIONS

The following equipment is required for online applications:

- 6/34 or 6/36 Central Processor with at least 8K words of memory.
- Program load device: diskette or cartridge disk attached through a Multiple Device Controller or Mass Storage Controller, respectively. An ASR-33 can be used as a load device. However, loading an online application from paper tape is slow. A card reader is also available, but the card loader can be used only for specially-supplied Honeywell-written applications.
- A console device is desirable but not required; the console can be attached through a Multiple Device Controller or a Multiline Communications Processor.

PERIPHERAL AND COMMUNICATIONS EQUIPMENT

Table B-1 lists the peripheral and communications equipment that can be used.

Table B-1. Usable Peripheral and Communications Equipment

Type Number	Description
DIU9101	Single diskette
DIU9102	Dual diskette
CDU9101	Cartridge disk, low density, removable disk (1.25 million words)
CDU9102	Cartridge disk, low density, fixed and removable disk (2.5 million words)
CDU9103	Cartridge disk, high density, removable disk (2.5 million words)
CDU9104	Cartridge disk, high density, fixed and removable disk (5.0 million words)
CRU9101	Punched card reader, 300 cpm
CRU9102	Punched and marked card reader, 300 cpm
CRU9103	Punched card reader, 500 cpm
CRU9104	Punched and marked card reader, 500 cpm
CRF9101	51-column card option for CRU's 9101, 9102, 9103, 9104
PRU9101	Serial printer, 60 lpm, 64-character set
PRU9102	Serial printer, 60 lpm, 96-character set
PRU9103	Line printer, 240 lpm, 96-character set
PRU9104	Line printer, 300 lpm, 64-character set
PRU9105	Line printer, 480 lpm, 96-character set
PRU9106	Line printer, 600 lpm, 64-character set
PRF9102	12-channel vertical format unit option for PRU's 9103, 9104, 9105, 9106
TWU9101	Typewriter console, 30 characters per second
TTU9101	Teleprinter console (ASR-33)
TTU9102	Teleprinter console (KSR-33)
DKU9101	CRT/keyboard console, 64-character set
DKU9102	CRT/keyboard console, 96-character set
MLC9101	Multiline Communications Processor, with Communications-Pac for eight asynchronous lines
MLC9102	Multiline Communications Processor, with Communications-Pac for eight synchronous lines
MLC9103	Multiline Communications Processor only – requires Communications-Pac(s) depending on choice of line speeds
DCM9101	Communications-Pac, two asynchronous lines, with cable
DCM9102	Communications-Pac, one asychronous line, with cable
DCM9103	Communications-Pac, two synchronous lines, with cable
DCM9104	Communications-Pac, one synchronous line, with cable

APPENDIX C
GLOSSARY

address, absolute

A nonrelocatable reference to a storage location.

algorithm

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

application configuration

The process by which user-written programs are combined with BES software under the direction of the Configuration Load Manager to produce an online application.

application program

User-written business, industrial, or scientific program.

argument

An independent variable that is passed to a system program or subroutine.

ASCII (American Standard Code for Information Interchange)

This is the code established as an American Standard by the American Standards Association.

ASR

Automatic send-receive unit. A combination teleprinter receiver and transmitter with transmission capability either from the keyboard or paper tape, and reception capability to paper tape or printer. See KSR.

attach table

A structure built by the Command Processor from information supplied to it in attach (AT) commands. Contains file information for programs to be executed.

backpatching

A software technique that resolves forward and/or external references within programs.

Binary Synchronous Communications (BSC)

Uniform procedure, using a standardized set of control characters and control character sequences, for synchronous transmission of binary-coded data between stations in a communications system.

bootstrap routine

A routine, contained in a single record, that is read into memory by a ROM bootstrap loader (see below), which can read a complete loader program into memory.

breakpoint

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

buffer

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

BTT

Basic timer table. A 16-word control block containing the four basic timers, their reset values, and pointers to the connected clock timer blocks (CTB's). Resides within the Clock Manager module.

byte

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

calling sequence

Standard code sequence by which system services or external procedures are requested.

checksum

An optional verification technique for paper tape read and punch operations. During a punch operation, the checksum computation consists of adding each pair of bytes of user text into a word area that has been initialized to zero. The overflow from any carry is added to the least significant position. The checksum value is escaped (see below), if necessary, and punched out following the RS delimiter after the related text bytes.

During a read operation, the checksum value is restored, if it had been escaped, before the checksum comparison is made. An error is reported if the checksum value does not agree with the computed value, or if the checksum value is missing.

clock frequency

The line frequency, in cycles per second, that is the basis (coupled with the scan cycle) for calculating the real-time clock-generated interrupts. Possible values are: 50 or 60 Hz. The default value is 60 Hz (U.S. standard).

clock scan cycle

The time in milliseconds between clock-generated interrupts. Possible values depend on the line frequency, and are integral multiples of 8.3 ms at 60 Hz or 10 ms at 50 Hz.

command input device

Any device used to submit commands to system programs and Configuration Load Manager.

communications device

A device that transfers data over communications lines and is connected through the MLCP (e.g., KSR-like terminals, VIP 7700).

console

Any of the following peripheral devices attached through the MDC: KSR teleprinter, ASR teleprinter (when used without paper tape), CRT console, and typewriter console.

CTB

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

device driver

A software component that performs all data transfers to and from a specific type of 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 relative file by supplying its relative record number.

directory record

A record in the volume directory containing a description of a file.

disk

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

displacement

A value that is added to the contents of an address register to give the effective address.

DDB

Directory descriptor block. An FDB (see entry below) describing a directory file.

dynamic file

A file that is created during online execution. Dynamic files can be expanded, contracted, or deleted during execution.

EOQ header table

End of queue header table. This table contains pointers to the last block in each task request queue.

escaping

The process of altering paper tape special characters that appear in the data portion of a message text so that they do not perform their usual control functions. The process involves performing an exclusive OR of the special character against the hexadecimal value 30 for encoding (escaping) as well as decoding (restoring) the character. During the punching operation, the escaped character is preceded by an escape control character (the hexadecimal value 7C) so that when the data is read, the control character will trigger the restoring of the next character to its original value.

extent

Contiguous allocated space on a disk.

external procedure

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

FCB

File control block. A data structure constructed by the Configuration Load Manager for use by the File Manager. An FCB is pointed to by an entry in the logical file table, and in turn, points to a file descriptor block.

FDB

File descriptor block. A data structure constructed by the Configuration Load Manager for use by the File Manager. An FDB is pointed to by an FCB, and in turn, points to a volume descriptor block for a particular file.

file name

A 1- to 12-character name assigned to a collection of related data records. For a file on disk this name is assigned when the file is statically allocated or dynamically created. For a nondisk file, this name is assigned by the DEVFILE command to the Configuration Load Manager. See "pathname" below.

FRIOR

FORTRAN Run-Time Input/Output Routines. A set of reentrant routines that work in conjunction with the File Manager to perform device manipulation, data transfer and processing for FORTRAN object programs.

function

A procedure that returns a single value to its caller. Contrast with "subroutine."

GPDMAI

General purpose direct memory access interface. A programmable controller that permits attachment of non-Honeywell peripheral devices.

HMA

High memory address.

initial system console

The console assigned as the system console during bootstrapping and loading; this device is the one through which the operator initially communicates with the system program that has just been loaded (see also "system console").

input/output device

A peripheral or communications device.

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 to, or asynchronous with, the executing program.

interrupt vector

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

IORB

Input/output request block. A control structure used for communication between a program and an I/O driver.

ISA

Interrupt save area. An area used to store the context of an interrupted task. There is one ISA for each priority level in use.

KSR

A keyboard send-receive teleprinter.

KSR-like terminal

A KSR teleprinter or CRT keyboard terminal that transfers data over communications lines and is connected to the MLCP.

LFN

Logical file number. A number that becomes associated with a file when it is opened. LFN's are used in all references to the file until it is closed.

LFT

Logical file table. A data structure constructed by the Configuration Load Manager for use by the File Manager. It contains an entry for each logical file number.

loader

Device-specific software component that loads programs into memory from disks, cards, or ASR paper tape.

loader communication area

An area within the loader in which key information is stored for the subsequent loading of the various system programs.

load module

A program, or portion of a program, that can be loaded as a single entity by a loader. Load modules are produced by the Linker and are stored as individual members of partitioned files.

LRN

Logical resource number. A number assigned to either a task or a device, and associated to a given priority level.

LRT

Logical resource table. A data structure built by the Configuration Load Manager and containing an entry for each logical resource number used in a configured application.

MBZ

Must be zero.

MDC

Multiple device controller for peripheral devices other than cartridge disk.

member

A source, object, load, or list text module that exists on a disk as a subfile of a partitioned file.

member name

A 1- to 8-character alphanumeric identifier of a member of a partitioned file.

MLCP

Multiline communications processor.

MSC

Mass storage controller for cartridge disks.

multi-extent file

A file consisting of more than one extent, possibly noncontinguous.

[illegible]

Next available trap save area pointer.

object module

A relocatable program unit produced by a single execution of the FORTRAN or COBOL Compilers, or the Assembler, and requiring further processing by the Linker before it is executable. Object modules are stored as individual members of partitioned files.

offline

An execution environment not controlled by BES Executive software.

online

An interrupt-driven application execution environment controlled by BES Executive software.

operator's console

A console specified for use in interactive communication between the operator and Honeywell-supplied software; used with Configuration Load Manager and online applications, (see also "system console"). For online applications, a KSR-like communications terminal can be assigned as the operator's console.

overlay

A section of a program that can be loaded during execution to overlay an area of memory. Used when there is insufficient memory to accommodate all the code of a program.

partitioned file

A file composed of individually accessible subfiles (members) with unique member names. Partitioned files are used to store programs as source, object and load modules.

patch

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

pathname

A character string that identifies a file when it is opened. The pathname for disk files consists of a 1- to 6-character volume name, followed by a mandatory "greater than" sign, followed by a 1- to 12-character file name. The pathname for nondisk files consists of a 1- to 12-character file name.

peripheral device

A noncommunications device connected through the MDC or MSC (e.g., card reader, console, disk).

PPT

Pool parameter table. A structure built by the CLM for use by the Buffer Manager.

priority level

Refers to the 64 numeric values that may be assigned to tasks and devices 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.

priority level activity indicators

Each bit of a 4-word area in memory is used to indicate whether or not a task is active at that level. When the bit is "off," no task is active at that level.

program counter

The P-register. Contains the address of the current instruction.

program name prefixes

BES system software modules are named using a standard 2-character prefix. The first character is a "Z"; the second character is unique and defines the major category to which the module belongs, e.g., "X" for Executive modules, "I" for I/O modules, and so on.

program name suffixes

BES system programs assign specific suffixes to the programs they process, so that it is possible to distinguish among the various source, object, list, and load modules on a library. The suffix is a "point-letter" such as ".O" for object modules, ".A" for assembly language source modules, and so on.

range

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

RCT

Resource control table. A control structure created by the CLM for use by the Task Manager to control task processing.

reentrant routine

A property of a routine that during execution does not alter itself; thus a reentrant routine can be entered and reused at any time.

relative file

A disk file that is logically divided into fixed-length records, where each record can be identified by its position relative to the beginning of the file. It may be accessed directly by record number, or sequentially.

relative record number

A number representing the position of a record relative to the beginning of a file. The initial record is relative record number 0.

relative sector number

The position of a sector relative to the beginning of a disk volume. For example, the relative sector number of the first sector on track 2 of a diskette is 52. The initial sector of a volume is relative sector number 0.

relocation factor

A number that is added to the origin of a program to provide the loading address for the program. The relocation factor is also added by the loader to every relocatable address within the program.

remote-extent record

A record in the volume directory pointing to extents of a multi-extent file.

request block

See IORB and TRB.

request queue

A threaded list of request blocks.

request queue header table

See "EOQ header table" and "SOQ header table."

return address

The address of the instruction in a program to which control is returned after a branch 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 input device into memory.

RSU

Reserved for system use.

sector

A 128-byte portion of a diskette track or 256-byte portion of a cartridge disk track.

sequential access

The method for reading or writing a record in a file by requesting the next record in sequence.

shareable file

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

SOQ header table

Start of queue header table. This table contains pointers to the first block in each task request queue.

source module

A program written in a source language for processing by a compiler or an assembler. Source modules are stored as individual members of partitioned files.

start address

The labeled address in a program at which execution begins.

static file

A file for which disk space has been allocated by using the allocate utility. Static files are not expandable or shrinkable during execution but can be deleted.

subroutine

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

system console

A console through which the operator communicates with a system program. The system console can be reassigned to a like device through an EX command to the Command Processor.

system program

Any of the program development tools or offline utility programs.

task

A task may be characterized as the execution of a sequence of instructions that has a starting and ending point and performs some identifiable function. Multiple tasks operate independent of and asynchronous to each other, but synchronize their operation by having one task either initiate another task for execution, wait for the completion of another task, or terminate itself. These tasking operations are obtained by calling task management functions.

transparent mode transmission

In this mode, data consisting of bytes having any bit configuration, even one that would usually affect the communications controller, can be transmitted over communications lines.

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.

TRB

Task request block. This data structure is used by one task to request another task and communicate with it.

TSA

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.

VDB

Volume descriptor block. A control structure built by the CLM for use by the File Manager. It contains information about the volume directory and volume contents.

volume allocation bit map

A control structure on every disk volume that indicates sector usage on the volume.

volume directory

A control structure on every disk that carries information about all the files on the volume.

volume index of defective sectors

A control structure on every disk volume that indicates which, if any, are the defective sectors on the volume.

volume label

A control structure on every disk volume that contains the volume identifier and other characteristics of this volume.

volume relative sector number

See "relative sector number."

INDEX

HONEYWELL INFORMATION SYSTEMS
Technical Publications Remarks Form

TITLE	SERIES 60 (LEVEL 6) GCOS/BES2 SOFTWARE OVERVIEW AND SYSTEM CONVENTIONS	ORDER NO.	AU50, REV. 0
		DATED	JULY 1976

ERRORS IN PUBLICATION

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

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

FROM: NAME ______________________ DATE ____________

TITLE ______________________

COMPANY ______________________

ADDRESS ______________________

____

FIRST CLASS
PERMIT NO. 39531
WALTHAM, MA
02154

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

Postage Will Be Paid By:

HONEYWELL INFORMATION SYSTEMS
200 SMITH STREET
WALTHAM, MA 02154

ATTENTION: PUBLICATIONS, MS 486

Honeywell

Honeywell

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

16343, 1876, Printed in U.S.A.

AU50, Rev. 0