MP/AOS

Concepts and Facilities

DataGeneral

Notice

Data General Corporation (DGC) has prepared this document for use by DGC personnel, customers, and prospective customers. The information contained herein shall not be reproduced in whole or in part without DGC's prior written approval.

DGC reserves the right to make changes in specifications and other information contained in this document without prior notice, and the reader should in all cases consult DGC to determine whether any such changes have been made.

The terms and conditions governing the sale of DGC hardware products and the licensing of DGC software consist solely of those set forth in the written contracts between DGC and its customers. No representation or other affirmation of fact contained in this document including but not limited to statements regarding capacity, response-time performance, suitability for use or performance of products described herein shall be deemed to be a warranty by DGC for any purpose, or give rise to any liability of DGC whatsoever.

In no event shall DGC be liable for any incidental, indirect, special or consequential damages whatsoever (including but not limited to lost profits) arising out of or related to this document or the information contained in it, even if DGC has been advised, knew or should have known of the possibility of such damages.

NOVA, INFOS, ECLIPSE, ECLIPSE MV/8000, DASHER, microNOVA, ENTERPRISE, and **PROXI** are U.S. registered trademarks of Data General Corporation, and **AZ-TEXT, DASHER, DG/L, ECLIPSE MV/6000, METEOR, PRESENT, REV-UP, SWAT, XODIAC, GENAP,** and **TRENDVIEW** are U.S. trademarks of Data General Corporation.

Ordering No. 069-400200

Rev. 00, April 1982

Preface

This manual is written to familiarize original equipment manufacturers and experienced system programmers with MP/AOS capabilities.

The book is divided into four chapters. Chapter 1 gives an overview of MP/AOS features. Chapter 2 describes the MP/AOS operating system. Chapter 3 contains a summary description of each utility. Chapter 4 deals with the programming languages supported by MP/AOS. The book also contains an index listing the concepts and terms described in Chapters 1 through 4; a listing of DG offices worldwide; a form explaining how to order technical publications; a Technical Products Publications Comment Form; and a Users' Group Membership Form.

The following manuals also belong to the series of books published on the MP/AOS operating system.

MP/AOS System Programmer's Reference (DGC No. 093-400051) documents MP/AOS system structure and provides a complete dictionary of system calls and library routines.

MP/AOS Command Line Interpreter (CLI) (DGC No. 069-400201) describes the interactive CLI program, the user's primary interface to the MP/AOS system. A command dictionary provides command descriptions, formats, and examples.

Loading MP/AOS (DGC No. 069-400207) describes how to install MP/AOS software on any ECLIPSE-line computer and how to load tailored systems.

MP/AOS System Generation and Related Utilities (DGC No. 069-400206) describes the generation of an MP/AOS system tailored to specific applications. It also describes the following utilities, including sample dialogues as appropriate:

- SYSGEN, the interactive system generation utility;
- DINIT, the disk initializer;
- FIXUP, the disk repair utility;
- SPOOLER, which controls line printer operations;
- ELOG (error logger), the utility for interpreting the system log file.

MP/AOS Debugger and Performance Monitoring Utilities (DGC No. 069-400205) describes the following utilities, providing a dictionary of debugger commands and sample dialogues as appropriate:

- FLIT, the process debugger;
- PROFILE, which measures execution-time performance;
- OPM, the process monitor that displays current system resource allocation and status.

MP/AOS Macroassembler, Binder, and Library Utilities (DGC No. 069-400210) documents the MP/AOS macroassembler and binder as well as the library file editor (LED) and system cross-reference analyzer (SCAN). It includes programming examples and a dictionary of assembler pseudo-ops.

MP/AOS Advanced Program Development Utilities (DGC No. 069-400208) describes the following utilities:

- Text control system (TCS), a method for managing different versions of a single file;
- BUILD, which creates a new version of a file from existing files, thus minimizing effort and errors in program development;
- FIND, which identifies occurrences of strings in text files.

MP/AOS SPEED Text Editor (DGC No. 069-400202) documents the features of SPEED, the MP/AOS character-oriented text editor.

MP/AOS SLATE Text Editor (DGC No. 069-400209) documents the features of SLATE, a screen- and line-oriented text editor.

MP/AOS File Utilities (DGC No. 069-400204) describes the following utility programs, providing sample dialogues for each:

- FEDIT, a file editor that permits modification of system files, including program and data files;
- FDISP, which can display the address and data contents of a file or compare two files, displaying the parts that differ;
- SCMP, which can compare two source programs line by line;
- MOVE, which allows the transfer of files among directories;
- AOSMIC, which allows manipulation of MP/AOS and MP/OS disks and files on an AOS system;
- FOXFIRE, which permits the transfer of files among MP/OS, MP/AOS, and AOS systems over asynchronous communication lines.

SP/Pascal Programmer's Reference (DGC No. 069-400203) documents an extended Pascal for system programmers. SP/Pascal has all of the features of MP/Pascal as well as special features targeted for the MP/AOS and AOS operating systems.

Books on three additional programming languages supported by MP/AOS have previously been published as part of the bookset for the MP/OS operating system:

MP/Pascal Programmer's Reference (DGC No. 069-400031) documents for system programmers a Pascal-based language, targeted for the MP/OS operating system.

MP/FORTRAN IV Programmer's Reference (DGC No. 069-400033) documents for system programmers a language based on ANSI 1966 standard FORTRAN with extensions.

MP/BASIC Programmer's Reference (DGC No. 069-400032) documents for new users a programming language based on ANSI standard BASIC with extensions.

MP/OS

For information on Microproducts and a bibliography of documentation on the Microproducts line, see *Introduction to Microproducts* (DGC No. 014-000685).

For information on cross development between MP/OS and MP/AOS, see *MP/OS System Programmer's Reference* (DGC No. 093-400001).

Contents

Figures

Tables

Overview and Summary

The MP/AOS system is a general-purpose operating system for 16-bit Data General ECLIPSE processors available in both 15-inch and 7 x 9-inch packaging. Supported processors must have the floating point instruction option and the character instruction set. Currently (March, 1982), MP/AOS runs on the following ECLIPSE processors: S/20, S/120, S/130, S/140, C/150, C/330, S/250, C/350.

MP/AOS retains substantial compatibility with MP/OS, a single-user system for the microNOVA, ENTERPRISE, MPT, MBC, and NOVA 4 computers. And with the aid of the System Call Translator software package, MP/AOS programs can be developed and run under AOS, the Advanced Operating System for the ECLIPSE line of computers.

MP/AOS provides sophisticated facilities such as multiprogramming, multitasking, interprocess communication, flexible user management of system resources, the ability to access more than 32 Kwords of memory from within a single user process, and a debugger as a separate process. The system features fast switching of task and process, deterministic scheduling of process/task strictly by priority, a time-slicing option for process scheduling, and low interrupt latency.

MP/AOS also supports user written device drivers, and provides system calls allowing users to define and manage custom debugger and histogrammer programs.

The MP/AOS operating system can be used to provide a basic program development environment; to that end, a full range of program development utilities, text editors, and high-level languages such as MP/FORTRAN IV, MP/ and SP/Pascal, and MP/BASIC is made available.

Additionally, MP/AOS is designed to provide an efficient basis for user-designed applications such as real-time process control, data acquisition, and medical instrumentation. Features such as memory resident processes, highly accurate timing (including the ability to time to milliseconds when a 1000-Hz clock frequency is selected at system generation), task synchronization, nonpended system calls, extended I/O with direct transfer of data between a device and memory, and support for custom devices make MP/AOS particularly well suited to real-time applications.

Using an interactive system generation utility, you can generate an MP/AOS system containing a desired subset of the full system's power, and tailor it to any configuration of memory boards and peripherals.

The user communicates with the operating system via a wide variety of *system calls*, command macros that call on predefined system routines. In assembly language, system calls are coded in the user program just as instructions are. Some calls have *options* modifying the call's action; options are specified by a two- or three-letter abbreviation coded after the call mnemonic in the program. After executing a system call, MP/AOS passes control to an *error return* or to a *normal return*, depending on the call's outcome.

This manual distills MP/AOS capabilities for the system OEM and the experienced system programmer. Readers not familiar with DGC products will find here a concise introduction to MP/AOS concepts and facilities, enabling them to assess the system's advantages.

Full documentation for MP/AOS is listed in the Related Manuals section preceding this chapter.

For those readers already familiar with other DGC systems, in particular with the sophisticated features of the Microproducts Operating System (MP/OS), this chapter summarizes comparative information and pinpoints the major advances embodied in the MP/AOS system.

Utilities

Several useful utilities are included as part of the MP/AOS system. For an overview, these utilities are grouped by function and listed in Table 1.1. A summary description of each utility appears in Chapter 3.

Programming Language Support

MP/AOS supports the MP/FORTRAN IV, MP/Pascal, SP/Pascal, and MP/BASIC programming languages. A brief description of each appears in Chapter 4.

Utility	Function
Program Development Utilities	
Command Line Interpreter (CLI)	Primary interface between the MP/AOS system and the user; supports macro command file facility with argument parameters.
Speed Text Editor (SPEED)	Character-oriented text editor with macro facility.
Slate Text Editor (SLATE)	Screen-oriented text editor with key definition macro facility.
Macroassembler (MASM)	Produces object file; translates assembly language source code and expands macro calls.
Binder (BIND)	Produces an executable program file.
Library Editor (LED)	Creates a library file consisting of one or more object modules.
Symbol Cross Reference Analyzer (SCAN)	Provides information about the use of global symbol names in libraries and object modules.
Histogrammer (PROFILE)	Measures and reports execution-time performance of a user process.
Process Debugger (FLIT)	Runs as a separate process; aids in detecting and correcting runtime errors in executing programs.
Process Monitor (OPM)	Provides a display of current system resource allocation and status.
Text Control System (TCS)	Maintains multiple versions of a file.
Build program (BUILD)	Using TCS, selects the correct version of a file for processing.
Find program (FIND)	Examines a text file for the occurrence of a pattern.

Table 1.1 MP/AOS utilities

Utility	Function
File Management Utilities	
File Editor (FEDIT)	Allows examination and modification of the files in an MP/AOS system.
File Display and Comparison (FDISP)	Compares two nontext files and displays differences in a variety of formats.
Source Compare Program (SCMP)	Compares two source programs line by line; indicates changes needed to transform the first file to the second.
Move Utility (MOVE)	Allows transfer of file copies between directories, as well as between disk/diskette and magnetic tape devices; AOS dump compatible format.
AOS File Transfer Utility (AOSMIC)	Allows AOS files to be moved to and from an MP/AOS disk (runs under AOS).
File Transfer Utility (FOXFIRE)	Permits the transfer of files between MP/OS, MP/AOS, and AOS systems over asynchronous communication lines.
System Management Utilities	
System Generation (SYSGEN)	Allows generation of MP/AOS systems tailored to user needs.
Disk Initializer (DINIT)	Software formats MP/AOS disk(ettes) and installs system files on them.
Disk Repair Program (FIXUP)	Repairs defective disk structures.
Spooler Utility (SPOOLER)	Controls the queueing and printing of files at line printers.
Error Logger (ELOG)	Logs error reports during system execution.

Table 1.1 MP/AOS utilities (continued)

Compatibility with AOS & MP/OS

MP/AOS is designed for compatibility with Data General's Advanced Operating System (AOS) and Data General's Microproducts Operating System (MP/OS). Programs prepared under AOS or MP/OS can be used under MP/AOS with minor modifications. For use on MP/AOS, program files must be rebound and some files may have to be reassembled. Data files need not be changed.

MP/AOS disk format is compatible with MP/OS. Hence, an MP/OS disk containing a file system may be used under MP/AOS without modifications.

MP/AOS is bind-time compatible with all MP/OS features, except for user device interrupt handlers, which need reassembling with minor modifications.

Relationship to MP/OS

MP/AOS extends the MP/OS operating system to accommodate a multiprocess environment, user management of memory beyond the 32-Kword user address space, user access to the data channel and BMC maps, and I/O to extended memory. These extensions have been implemented through the addition of new calls and, in many cases, through new options to existing calls. For the benefit of readers already familiar with MP/OS, this section summarizes the main differences between MP/AOS and MP/OS.

The reader new to DGC products may wish to skip this section turning directly to Chapter 2 where the main features of the MP/AOS operating system are summarized.

Memory Protection and Process Security

Under MP/AOS, user processes run within their own mapped address space. Hence, under normal operation, user processes are incapable of accessing, manipulating, or destroying the operating system, or each other.

The mapped environment offers several advantages, notably:

- support for a larger physical address space;
- more memory available for user access;
- the user process does not need to share its 32-Kword memory area with the system;
- the operating system is protected from the user;
- users are protected from each other.

MP/AOS system memory is organized into a *mapped supervisor* program and an *unmapped* portion, known as the *kernel.*

MP/AOS main memory is available to a user process in logical allocations of *pages* (blocks of 1 Kwords). One or more pages are grouped into units called *segments.* Each user process has a maximum logical address space of 32 Kwords, allocated to three segments corresponding to *impure*, *shared*, and *overlay* memory.

Impure memory is unshared, containing information used only by one program. *Shared* memory contains information which can be shared among programs executing in different processes. *Overlay* memory is the area reserved for routines which are only read into memory when the program actually needs them. Together, the shared and overlay memory areas constitute process *pure* memory.

A program executing within a process logical address space can define, attach, and map to additional memory segments as needed. Programs executing within *different* processes can attach to the same memory segment, thus sharing impure data.

The MP/AOS use of dynamic memory segments with the hardware mapping feature enables a program effectively to address all of physical memory not used by the operating system, provided the addressing extends to no more than 32 Kwords of physical memory at any one time.

Overlays

As in MP/OS, overlay loading and release are accomplished with library routines bound into the user address space. Certain MP/OS restrictions on overlay use across program swaps are relaxed in MP/AOS.

Program/Process Management

MP/AOS supports a *multiprogramming* environment in which users can execute several processes in parallel and independently of each other. Although any process can create other processes, no hierarchical relationship exists between the new process and the process creating it. All MP/AOS processes are *resident:* they are not dynamically swapped but reside in main memory until termination.

Within a process logical address space, the *execute/return* mechanism allows users to change their environment without losing their previous state in a particular program. Users can thus operate at different levels with different programs while remaining within the same process.

Swapping or *chaining* can be used to initiate new programs to run within the address space of a single process. When swapping (via the **?EXEC** system call), the state of the calling program is written (swapped out) to disk. The new program runs at the next higher *swap level* while remaining within the same process address space. When the new program terminates, the program on the next lower swap level is restored. The nesting of programs within the same process address space can continue for up to eight levels.

A *chain* operation (via the **?EXEC** system call with the chain option) changes the program a process is running while overwriting the calling program. Chained programs run on the same swap level as the program that initiates them.

Debugger

The MP/AOS debugger is not bound into the user program space. It functions as a separate process, taking control of the program to be debugged. This enables the user process to make full use of its 32-Kword address space even when debugging. Further, no preplanning is required to debug a program.

MP/AOS makes available system calls which enable users to write custom debugger programs.

Interprocess Communication

MP/AOS supports an Interprocess Communication facility, allowing processes to communicate with multiple, free-format messages of variable length across full-duplex communication *ports.*

Multitasking

The MP/OS concept of task priority is extended due to the fact that process priority supersedes task priority.

Task creation, termination, scheduling, and intertask communication are MP/OS compatible. Added options make some of these operations pertinent to processes.

File Management

The MP/OS mechanism for advanced file management is preserved intact under MP/AOS.

Input and Output

MP/AOS extends MP/OS read and write procedures with the addition of *extended* (direct segment) *I/O*; this mechanism permits direct data transfer between any memory segment and a disk device. If the memory segment is user-defined, it need not be mapped to user space.

User Device Support

MP/AOS capabilities permit user control of I/O protection for system and user devices, and support user-written device service routines, as well as user manipulation of data channel and burst multiplexor (BMC) map slots. Data channel or BMC mapping within a user interrupt routine is supported. Under MP/AOS, I/O enabling/disabling calls simultaneously affect LEF mode as well as the I/O protection bit.

Library Routines

Changes to the MP/OS library are minimal. Library routines for supporting data channel line printers have been added.

The MP/AOS Operating System

This chapter outlines the salient features of the MP/AOS system and its advantages to the user. The chapter covers memory management, multiprocess/multitask capabilities, interprocess communication process debugging, histogramming, file management, input/output, user device support, compatibility with AOS, and library routines.

Where appropriate, discussion sections are followed by tables of pertinent system calls. The material covered here is documented in the *MP/AOS System Programmer's Reference* (DGC No. 093-400051).

Memory Management

MP/AOS supports up to 1024 Kwords (2 megabytes) of memory, the maximum supported by ECLIPSE architecture.

The MP/AOS operating system makes use of the Memory Allocation and Protection (MAP) feature of the hardware. The MAP feature provides extended physical addressable memory for user programs together with various protection features for the operating system and for currently used program memory. Specific MAP features are processor-dependent and are detailed in the Principles of Operation manual appropriate to the given processor.

An MP/AOS program development system consists entirely of read/write memory (RAM). The MP/AOS scheme of memory organization and allocation offers the system designer significant flexibility in memory use. Control of memory resources leaves the user with access to all available memory, except that used by the system.

MP/AOS main memory is available to a user process in logical allocations of *pages* (blocks of 1 Kwords). One or more pages are grouped into units called *segments*.

Each user process has a maximum logical address space of 32 Kwords (65536 bytes). However, the use of dynamic memory segments with the segment remapping feature enables a process effectively to address all of physical memory, provided the addressing extends to no more than 32 Kwords of physical memory at any one time.

Available memory is acquired by the process itself in process-specific segment sizes when the process is created. A program executing within a process logical address space can define, attach, and map to additional memory segments when needed.

A program can share one or more memory segments with other programs.

The dynamic memory management of MP/AOS is handled through a number of system calls that allocate additional address space as needed and release unneeded address space for use by other processes in the multi-processing environment of MP/AOS.

The responsibility of memory management rests with those user processes sharing system resources at the time.

Memory Translation and Protection (MAP)

The MAP feature of the ECLIPSE hardware translates a 15-bit address into a 20-bit address with the aid of address translation hardware and the logical-to-physical address translation functions set up by the supervisor program. The resulting address is used to reference memory (see Figure 2.1).

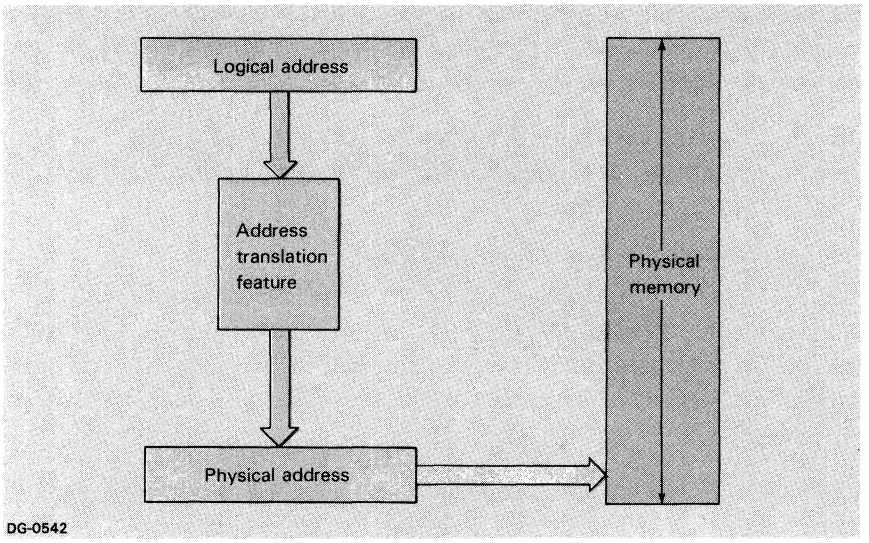

Figure 2.1 Logical to physical address translation

Main memory is allocated in *pages:* blocks of 1 Kwords (1024 2-byte words). The allocated blocks of physical memory do not have to be contiguous, and the MAP feature allows a different logical-to-physical address computation to be specified for each 1-Kword block of logical memory.

The ECLIPSE logical user address space is 32 Kwords, but with the MAP feature the maximum physical address space may be increased up to 1 megaword, depending on the machine used.

The address translation function which correlates a logical address to the corresponding allocated physical memory address is called an *address map.* MP/AOS supports seven address maps.

Two or more of these maps (depending on processor) are user address translation functions transparent to the user. Four of the maps are translation functions for the data channel, and one is a translation function for the burst multiplexor channel (BMC). Data channel and BMC maps can be manipulated by user-written device drivers (see "User Device Support," later in this chapter).

Map Protection Features

In addition to translating addresses, the MAP features also perform various protection functions. Four such functions are:

validity protection
write protection
indirect protection
I/O protection

Validity protection protects currently unused portions of the user process logical address space (up to 32 Kwords).

Write protection ensures that certain blocks of allocated physical memory will not be altered. Under MP/AOS, shared memory as well as overlay memory areas are write-protected.

Indirect protection is always enabled in the user process, ensuring that the system is not disabled by an indirection loop (an indirection chain exceeding 16 levels).

I/O protection controls the accessibility of I/O devices. Under MP/AOS, the I/O protection bit and Load Effective Address (LEF) mode are controlled simultaneously: when I/O is enabled, LEF mode and the I/O protection bit are simultaneously disabled, and vice versa. Initially, I/O protection and LEF mode are enabled in user processes.

The user can control I/O protection and the LEF mode by means of system calls.

Memory Organization for the System

In order to optimize system and interrupt performance, the MP/AOS system memory is divided into unmapped and mapped portions (see Figure 2.2).

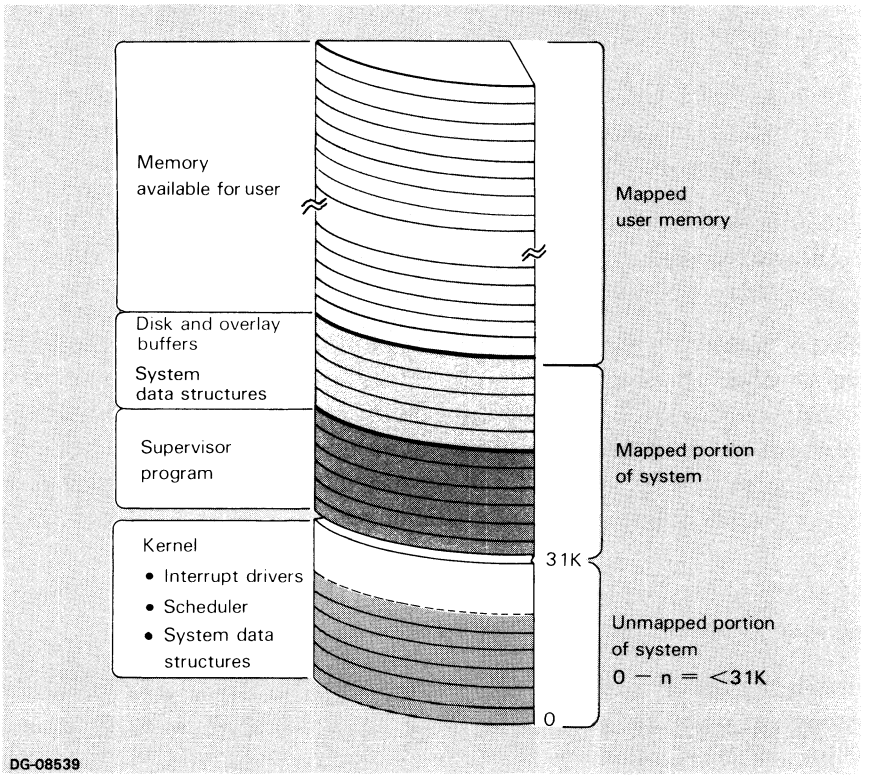

Figure 2.2 System memory configuration

The unmapped portion of the system is the *kernel.* It occupies physical memory lower page zero and may extend up to 31 pages,

depending on the type of system generated. The kernel contains the interrupt drivers, the scheduler, and major system data structures, as well as routines for their manipulation. Since the kernel runs unmapped, interrupt performance is maximized.

The mapped portions of the system consist of the *supervisor*, the disk and overlay *buffer areas (buffer cache)*, and some system data bases which are mapped as needed.

The *supervisor* program executes in mapped space. It is overlaid in order to minimize total physical memory requirements.

The supervisor program calls the kernel for the manipulation of data structures maintained by the kernel.

The preallocated disk and overlay buffer areas are dynamically mapped by the operating system.

User processes begin immediately after the disk and overlay buffer areas; these processes can utilize the rest of physical memory. User processes are always mapped.

Disk Buffering

To increase the efficiency of data transfer and to minimize disk access, MP/AOS provides a software-maintained *buffer cache* for disk I/O. These system buffers exist in neither the kernel nor the supervisor address space; rather, the system dynamically maps to them.

Although the size of the buffers is predetermined, their actual number is a system generation parameter. Depending on the demands placed on the system, these buffers are used to store either file system data or user data as needed.

In general, the system buffers are maintained on a least recently used (LRU) basis: the buffer most recently filled is last in line to be flushed to disk when new buffer space is required.

Typically, data in buffers is written to disk either when the disk buffer holding the data is needed to buffer other data, or when the disk file is closed. For applications where information on the disk file needs constant updates, MP/AOS provides a mechanism that allows data to be flushed from the buffers before the ?WRITE system call returns. For further discussion, see "Input and Output" later in this chapter.

The buffering mechanism is bypassed altogether for *block-aligned* data; that is, when the transfer request is for a byte count which is a multiple of the 512-byte sector size, when the destination address in memory is word-aligned, and when the file address is block aligned. In that case, data is moved directly from the device to the user

address space. In general, user data is buffered only for the minimum amount of data necessary; if part of the transfer can go directly, it will do so.

Organization of User Logical Address Space

MP/AOS allows the user to manage memory through calls which manipulate numbered regions called *segments.* A segment consists of one or more 1 Kword pages of memory.

Each user process is potentially capable of utilizing a logical address space of up to 32 Kwords. The actual amount of logical address space allocated is specified by the user when creating the process. (See "Process and Task Management" later in this chapter.) Memory fragmentation is avoided by the fact that the user program need not occupy physical locations that are contiguous.

The user process logical address space consists of *impure* and *pure* memory areas. *Impure* memory is unshared, containing information used only by one program. *Pure* memory can consist of two separate areas, namely, *shared* and *overlay* memory. *Shared* memory contains information which can be shared among processes running the same program. The sharing takes place automatically, in a manner transparent to the user. *Overlay* memory is the area in which the system places *overlay nodes.* (Overlay nodes are routines, or groups of routines, that are read into memory only when the process actually needs them.) The shared and overlay areas are both write-protected.

As Figure 2.3 illustrates, memory that corresponds to the user impure, shared, and overlay areas is allocated in three segments, locally numbered for each process as follows:

Segment 0 - Impure memory
Segment 1 - Shared memory
Segment 2 - Overlay memory

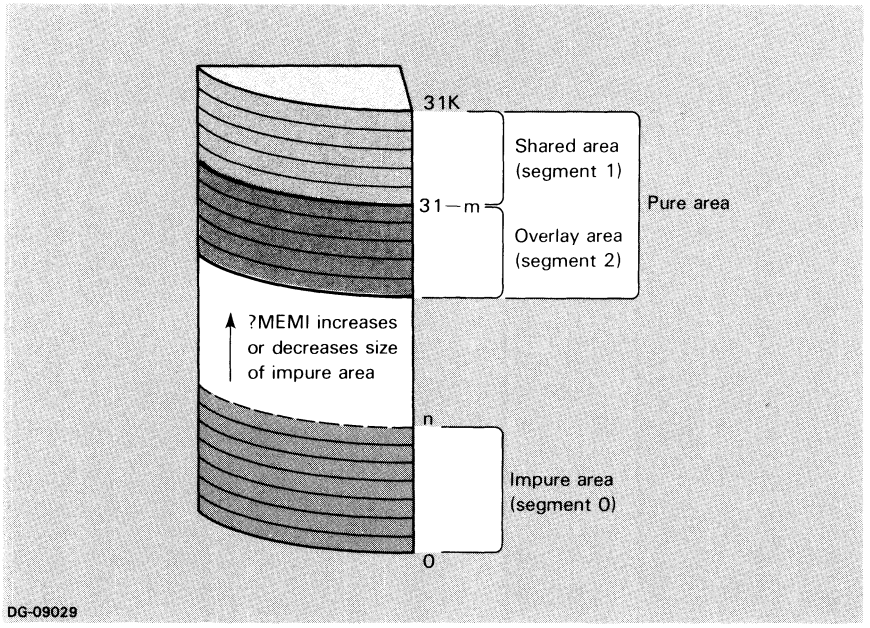

Figure 2.3 Organization of user logical address space

The local numbering acts as a shorthand reference for each process in dealing with its impure, shared, and overlay memory segments. Moreover, since locally numbered segments cannot be transmitted between processes, a certain measure of protection for user process memory is thereby achieved.

As shown in Figure 2.3, the size of pure memory (shared and overlay segments 1 and 2) remains fixed. By contrast, impure memory (segment 0) is *dynamic;* its size is allowed to grow and shrink within a maximum range determined by the size of the shared and overlay segments relative to the total size of memory specified by the user for that process.

Extended Memory Management

The dynamic segment facility allows for user management of additional memory areas added to the process after the initial or default segments are provided.

A program can, for example, define one or more additional memory segments, each of which contains one or more 1-Kword pages (though segment size is bounded), causing them to be attached to its address space, and releasing them at will. The maximum number of user-defined segments and of attached segments for all processes in the system is a system generation parameter.

A user program can also map desired portions of attached memory segments into user logical address space. This feature makes vastly enlarged memory resources available to the program. Optionally, the mapped pages can be write protected. Figure 2.4 illustrates segment mapping.

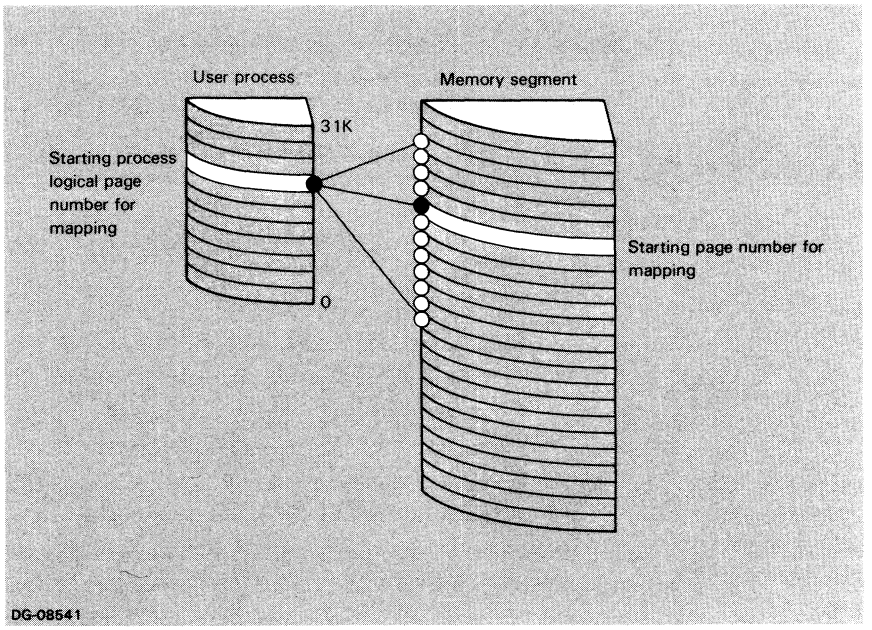

Figure 2.4 Mapping to a segment

Since any number of processes can attach to a segment once it has been created, the efficient passing of data between programs executing within different processes is greatly facilitated.

Another useful MP/AOS capability is the direct transfer of data between a device and a memory segment regardless of whether the segment is mapped to a given user address space.

A summary of memory-related MP/AOS system calls is shown in Table 2.1.

Call	Function	Option
?ASEG	Attach a memory segment	
?CSEG	Create a memory segment	
?DSEG	Detach from a memory segment	
?MEMI	Change impure memory allocation	
?MSEG	Map a memory segment	WP (write-protect mapped pages)

Table 2.1 Memory calls summary

Process & Task Management

A *process* is the system representation of a program with a unique identifier and a separate address space in some phase of execution. *Multiprogramming* presents the advantage of allowing the user to execute several processes concurrently and independently of each other.

A *task* can be defined as a separate flow of control within a particular address space. Multitasking simplifies certain types of programs, notably those which must perform a number of operations in parallel.

MP/AOS supports multiprogramming with extended capabilities that handle a number of simultaneous and independent processes. Capabilities provided for task management in a multitasking environment are extended to manage multiple processes. With minimized interrupt latency, rapid process switching, and the use of the ECLIPSE instruction set, MP/AOS provides the speed and performance to meet the requirements of process-control/real-time applications.

MP/AOS multiprogramming and multitasking capabilities enable development of a wide range of applications systems. Process and task servicing are user-defined with a priority system; extensive system calls provide for complete user control of process logical address space, process and task scheduling, and task I/O.

Process Concepts

The actual number of processes and tasks MP/AOS can support is a system generation parameter. This number is also affected by other system generation parameters requiring physical address space, such as the amount of space required for system data structures and for interrupt handling routines to service the number of devices specified relative to the size of the kernel. The current system generation limit is 16 processes.

When MP/AOS is initialized, the system creates a user process called the *initial process.* The specific program executed by the initial process is a system generation parameter; a user can specify any program file for this purpose. A typical program running as the initial process might be the Command Line Interpreter (CLI), the interface between the user and the system. Only the program running in the initial process is capable of issuing a system call to shut down the system.

A process can be created by any other process. MP/AOS processes are *not* hierarchical: they are created in parallel, executing *concurrently* within separate address spaces; once initiated, a process exists independently until it terminates itself or is terminated by another process, nor does it return to its creator upon termination.

All concurrent processes are also *resident,* that is, they reside permanently in main memory until their termination and cannot be displaced by other processes. MP/AOS does not support the swapping

of processes. On the other hand, a process represents an environment within which a program executes and which is capable of hosting a succession of programs; new programs may thus be initiated to run successively on different *swap levels* within the address space of a single process. See "Changing Programs within a Process."

Process Identifier (PID)

Once a process has been created, it is referenced by a unique *process identifier* (PID) generated by the system and ranging between 1 and 65535. Programs executing within processes can retrieve the process identifier of their own process, as well as that of any other process.

Process Priority

The system allocates CPU control to each process according to several parameters, one of which is process *priority*. The priority of a newly initiated process must be specified by the creating process but can subsequently be retrieved and modified by any program running within the new process. The value of the priority argument can range from 0 to 255, with 0 the highest priority and 255 the lowest. Programs can obtain the priority number of any user process in the system.

Since priorities are assigned to tasks as well as to processes, the concept of priority is a double one. For scheduling purposes, process priority has precedence over task priority. The highest priority process with ready tasks receives system services, regardless of the priority of its current task. A task is scheduled if it is ready to run (that is, not suspended for any reason such as waiting for I/O), and if it has the highest task priority *within the highest priority process*. Higher priority tasks within lower priority processes are ignored.

When generating an MP/AOS system, the user can choose *time-slicing* for processes of equal priority. When time-slicing is used, processes with the same priority number are scheduled in round-robin fashion, giving each process equal CPU time until all its tasks pend or until its time slice expires. Without time-slicing, round-robin scheduling for equal priority processes only comes into effect when control is relinquished by the process currently executing (when all its tasks pend) and not before. The time-slicing feature does not affect task scheduling.

Changing Programs Within a Process

MP/AOS offers the execute/return mechanism within a process logical address space. This enables users to change their environment without losing their previous state in a particular program. Thus, users can operate at different levels with different programs while remaining within the same process. The number of processes required for an application can therefore be reduced.

Either *swapping* or *chaining* can be used to initiate new programs to run within the address space of a single process. (The program executing within the initial process and the first program to execute within a new process run on the initial *swap level* of 1.) In a *swap,* the current state of the initiating (pushing), or *parent* program is swapped out (written to a disk file). Impure memory (segment 0), current segment mapping and relationships to it, and information needed for restoring overlays are among the items saved in the program state on a swap.

After the swap operation, the new program runs at the next higher *swap level* number, while retaining the same unique process identity number (PID) as the calling program.

When the new program terminates, the state of the parent program is restored ('popped'). The nesting of programs within the same process address space can continue for up to eight levels.

A *chain* operation changes the program a process is running, while retaining only the process *swap level* rather than the calling program state. Chaining overwrites the calling program with the new one without saving its state. The new program retains the same swap level as the initiating program. Figure 2.5 illustrates swapping and chaining.

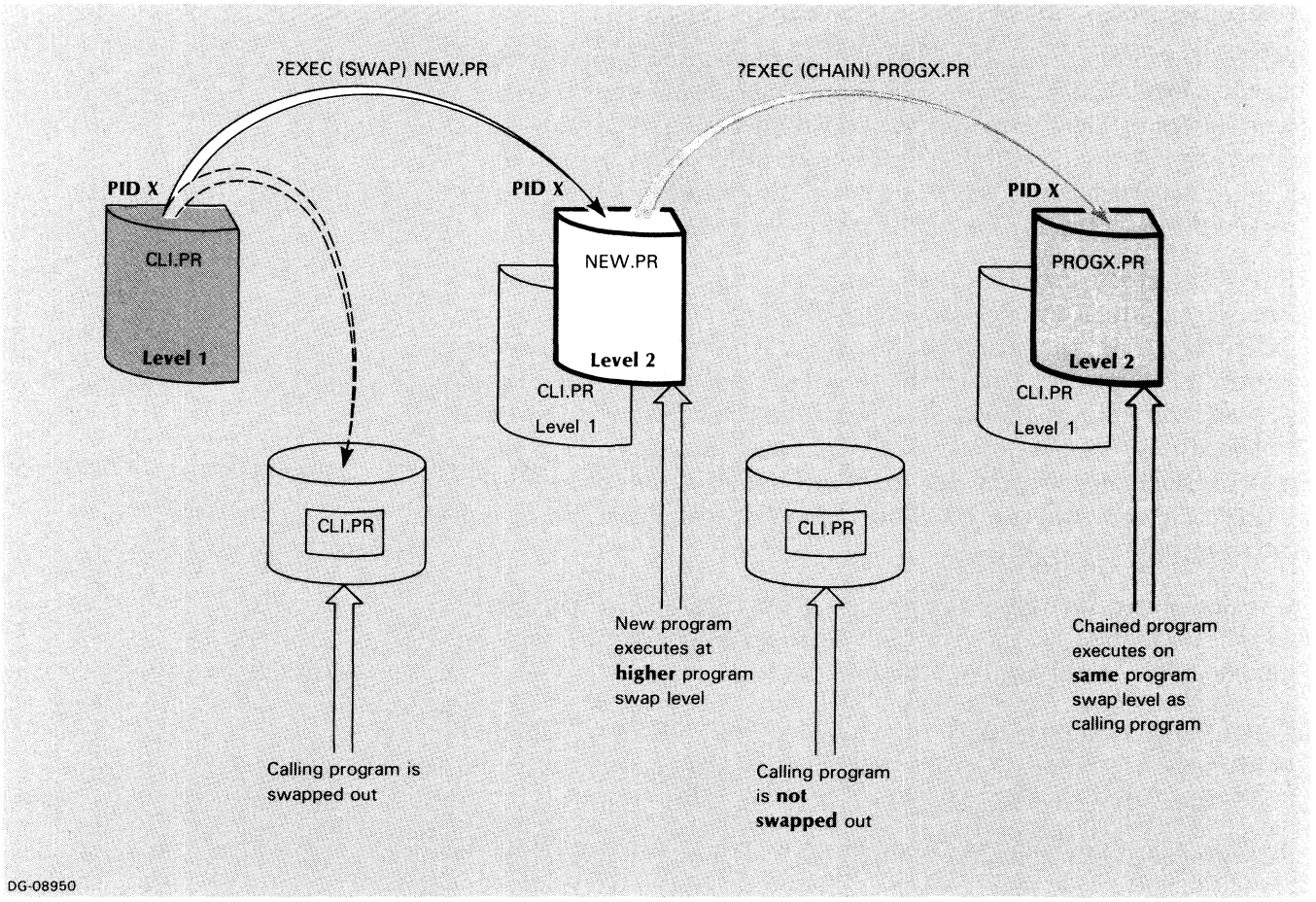

Figure 2.5 ?EXEC with swap and chain options

When a chained program terminates, it does not return to its calling program; instead, it reactivates the saved program at the previous swap level.

Managing Processes

Programs executing within a process logical address space can define their environment, and interrogate the system for information about a specific process and about system parameters. A program can also control process scheduling by blocking other processes, as well as the process within which it is executing, and by enabling and disabling the scheduler for other processes.

A program can terminate itself as well as programs executing within other processes. A message can be passed by a terminated program to the program which initiated it. (See "Changing Programs Within a Process".) The state of a terminated program can be examined if a *break* file is created from it at the time of termination.

A separate process debugger can initiate and block the process to be debugged. This feature prevents the loss of user process space which results if the debugger is bound in; it also allows debugging of the program, in its environment, without preplanning. MP/AOS also provides system calls for users wishing to write their own debugger programs.

Task Management

A given user program consists of one or more *tasks.* (A task, as previously stated, is defined as a separate flow of control within a particular address space.)

The total number of tasks MP/AOS can support is a system generation parameter. The current system generation limit is 255 tasks system-wide. Included in this number is a system task created for each process up to the maximum number of processes specified during system generation.

Since the system uses some memory to hold task control information, the user must specify the maximum number of tasks available to any single program executing within a process. This number, which is specified at the time the process is created, does *not* include the system-created task for each process.

Within their current program, tasks can control each other's activities by suspending and re-enabling task scheduling. They can also synchronize their activities to respond to external events and to exchange messages.

Task Identifiers

Tasks are referenced by a *task identifier* (TID) assigned by the system when the task is created. Tasks can retrieve their own task identifiers with a system call.

Task Priority

Tasks, like processes, are scheduled by *priority*. Task priority is specified by the user program upon task creation and can be modified at any time thereafter. Task priority is *subordinate* to process priority: tasks within the highest priority process are executed according to their relative priority. Tasks within lower priority processes are run only if no higher priority process has tasks which are ready to run.

Tasks can modify their own priorities, as well as the priorities of other tasks within their current program.

Nonpended Calls

MP/AOS provides a nonpended option for many system calls, notably Interprocess Communication and Input/Output calls. For many purposes this capability can serve as a useful variation of multitasking itself.

The nonpended option causes the system to create a new task which executes the nonpended call, while the task issuing the call can continue its operation. System tasks created in this fashion must be included in the total task number specified for the process.

The system allows the program to be notified when any one of its nonpended calls (tasks) has completed execution and to receive the identifier of the completed task. The nonpended call's outputs can then be ascertained.

MP/AOS process and task management system calls are summarized in Table 2.2.

Call	Function	Options
?BLOCK	Block a process	
?BOOT	Shut down the system	
?CTASK	Create a task	AW (await TCB if none available)
?DRSCH	Disable task or process rescheduling	CK (take error return if multitasking already disabled) PRC (act on process)
?EINFO	Get process information	
?ERSCH	Enable task or process rescheduling	PRC (act on process)
?EXEC	Execute a program	CL (close all channels except for the standard console I/O channels)
?GTPID	Get Process IDs of all processes in the system	
?IFPU	Use floating point unit	
?INFO	Get program information	PID (return process information)
?KILL	Terminate a program	BK (create a break file)
?KTASK	Terminate a task	
?MYID	Get task or process ID and priority	PRC (return process ID and priority)
?PEND	Suspend a task	
?PRI	Change task or process priority	PRC (act on process)
?PROC	Create a process	DB (debug)
?RETURN	Return to previous program	BK (create a break file)
?SINFO	Get information about the system	
?UNBLOCK	Unblock a process	
?UNPEND	Resume execution of a task	BD (unpend all tasks) ER (unpend at error return) ID (unpend on task ID, not event code)

Table 2.2 Process and task system calls summary

Interprocess Communication (IPC)

MP/AOS allows processes to communicate with one another using multiple, free-format messages of variable length. Processes send and receive messages using full-duplex communication *ports* and packets of specifications.

Processes can use the Interprocess Communication (IPC) facility in customer-server relationships whereby the server process is a central provider of certain functionality. Resource management, monitoring of other processes or the establishing of a network protocol are typical examples of server functions.

The IPC facility is also useful for synchronizing process activity, passing data between processes, or notifying other processes of execution results. Under MP/AOS, print commands can be passed from a program to the line printer utility SPOOLER through IPC calls. Inclusion of the IPC facility in an MP/AOS system must be specified by the user at system generation time.

The *customer-server* relationship is the mechanism for establishing a connection between two processes wishing to communicate. Once the server process has declared itself as a server, any number of customer processes can establish a connection with it by looking up its server name. Each customer process is returned a port number for conversing with that specific server. The two parties can then begin communicating.

A server process can receive messages over a general receive port, or it can post a receive call for messages from a particular customer port. This feature offers server and customer processes the option of communicating "privately" with each other.

MP/AOS supports multilevel connections, allowing a process to act as a server for some processes and as a customer of other processes.

Tasks sending and receiving messages can specify a *timeout* interval for the completion of the send or receive action: the calling task is then suspended for the specified time interval, unless the send or receive action is completed sooner.

Use of the *nonpended* option allows tasks issuing IPC calls to continue activity without delay. The nonpended option causes the system to create separate tasks to perform the IPC action; these system-created tasks are suspended until completion of the IPC call, while the tasks issuing the call continue normally.

IPC calls allow either party to send and receive messages, via specific ports, to break the customer-server connection, and to request each other's process ID (PID). Server-only calls allow the server process to receive messages over a general receive port, to invalidate its server function, to break off the connection between itself and all of its customers, and to obtain a list of all customers who have broken off connection with it.

Table 2.3 summarizes the Interprocess Communication system calls available under MP/AOS.

Call	Function	Options
?CLEAR	Clear IPC connection	
?DCLR	Declare server name	
?LKUP	Look up a server process and form connection	
?OBITS	Get list of broken connections	CK (don't wait)
?PURGE	Clear all connections to a server process	
?RCV	Receive data request	
?RCVA	Receive any request	NP (nonpended)
?RMVE	Server process name declaration removal	
?SD.R	Send/receive data request	NP (nonpended)
?SEND	Send data request	NP (nonpended)
?TPORT	Translate port to PID	

Table 2.3 Interprocess communication calls summary

File Management

A *file* is any collection of related data treated as a unit. Input and output *devices* are the means for physically storing and retrieving this information. One disk unit in every MP/AOS system is the system *master device*, the disk from which the system was bootstrapped; it contains the system program files and many other commonly used files.

MP/AOS file management facilities offer simple, efficient ways of communicating with multifile input/output devices and of storing and retrieving file data. The user view of the file system is compatible with the file systems of AOS and AOS/VS. The file system structure is identical to that of MP/OS.

All devices and files are handled by the same system calls, regardless of the physical device being read or written. This feature makes it easy to write device-independent programs.

In general, a file's contents are entirely user-defined; however, the system recognizes several types of files as having special functions. One important file type is the *directory*, which contains information about a specific set of files.

The MP/AOS system itself maintains a *device directory* — a table in memory whose filenames correspond to all the input and output devices in the system. Disk devices have a *root directory* that is the highest directory on the device containing the names of all files on that device. Figure 2.6 illustrates a hierarchical file system.

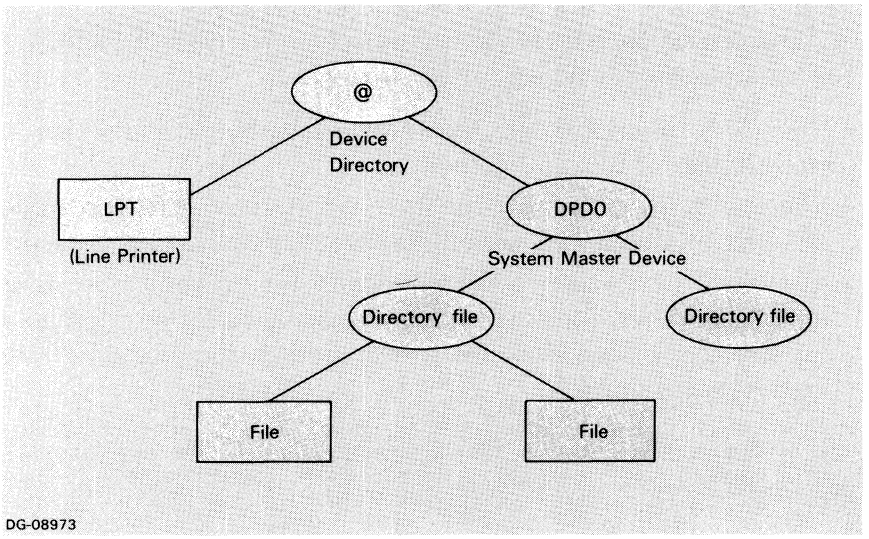

Figure 2.6 Hierarchical file system

Users can group related files into directories organized into a hierarchical structure with multiple levels of directory nesting for easy and logical file organization.

Files can be dynamically created and deleted. A time and date are associated with each file when it is created, and this information is updated when the file is accessed or modified. Users can assign their own file types to newly created files within a range of values reserved by MP/AOS for that purpose.

Disk space is allocated to a file based on the file's *element size*, which the user specifies when creating the file; elements are composed of 512-byte contiguous disk blocks. To keep track of each file's elements, MP/AOS maintains a hierarchical file index. This allows random access within files of varying element sizes.

MP/AOS offers file protection by means of a system-maintained *attribute* word for each file. The system and the user each has a portion of the attribute word reserved; hence users can set and modify file attributes for user-created files.

Files are referenced by means of *filenames.* The unique referencing of any file within a particular directory is accomplished by means of a *pathname* indicating a path through the directory structure to that specific file.

The system assigns to every program a *working directory* which may be thought of as the user's current location in the directory structure. The user program can designate any directory as its working directory and modify that designation at will.

Pathnames can be *fully qualified* (beginning at the device directory), or they can be *partial* for shorthand reference. Several techniques for streamlining file access are provided:

- *link files* containing part or all of a lengthy pathname can be substituted for that pathname;
- the *search list*, a user-modifiable list of directories to be searched for files, eliminates the need for typing a fully-qualified pathname;
- the *working directory*, which serves as a starting point of reference for file searches, allowing the search start to be defined in relation to directories superior or inferior to the working directory in the directory structure.

Table 2.4 details the file management system calls supported by MP/AOS.

Call	Function	Options
?ALIST	Alter searchlist	
?CREATE	Create a file	DE (delete existing file with same name)
?DELETE	Delete a file	
?DIR	Select a working directory	
?FSTAT	Get information about a file	CH (file is open on specified channel number) LNK (do not resolve links)
?GCHAR	Get device characteristics	CH (device is open on specified channel number) HC (return terminal's hardware characteristics) LL (return number of characters per line) PG (return number of lines per page) RS (return characteristics at time system was booted)
?GLIST	Get current searchlist	
?GNAME	Get fully-qualified pathname	CH (file is open on specified channel number) PID (get pathname of process) PR (get pathname of calling program)
?GTATR	Get file attributes and file type	CH (file is open on specified channel number) LN (return file byte length)
?RENAME	Rename a file	DE (delete existing filename)
?SCHAR	Set device characteristics	CH (device is open on specified channel number) HC (set terminal's hardware characteristics) LL (set number of characters per line) PG (set number of lines per page) RS (reset to boot-time value)
?STATR	Set file attributes	CH (file is open on specified channel number)

Table 2.4 File and device system calls summary

Input & Output

MP/AOS input/output capabilities include the following:

- user controlled allocation and release of I/O channels;
- file positioning for random access to disk files;
- the software maintained buffer cache;
- nonpended I/O;
- dynamic, block-aligned, and data-sensitive transfers;
- extended (direct segment) I/O, for direct data transfer between a user-defined memory segment and a disk device.

The following sections summarize these features and outline MP/AOS input/output device management.

I/O Channels

Data transfers between a program and a device take place over an *I/O channel* connected to the device by the appropriate system call. The maximum number of I/O channels that may be opened by a given program is specified by the user when creating the process within which the program executes. Up to 256 channels per process can be specified. The total number of I/O channels that may be opened system-wide is specified by the user at system generation time, up to a maximum of 512.

The user program controls the allocation and release of I/O channels.

An important feature of MP/AOS is its ability to pass I/O channels between programs executing within the *same* process. Optionally the calling program can close all but the standard I/O channels (see below) on behalf of the new program.

No channels are passed to newly created processes by the creating program.

Standard I/O Channels

The CLI always opens two channels for console I/O and always passes these two channels to other programs, closing all other channels before calling a new program.

When creating a new process, the user can specify alternate input and output devices to which the standard channels should be opened.

File Positioning

The system uses a 32-bit *file pointer* to track its position in an opened file. The pointer is at zero at the beginning of the file and is incremented for each byte transferred. The user program can ascertain the value of the file pointer and modify it for random access to any byte in the file.

I/O Buffers and the Flush Option

As previously discussed (see "Disk Buffer Area" in Memory Management section), data transfers are generally buffered through the system, thus providing efficient sequential I/O.

Under normal operation, the user is assured of a file update only when the disk buffer currently holding the data is needed for buffering other data, or when an I/O channel is closed (all program files are closed and buffers flushed on program termination); if the system were to crash before the file was closed, some data might be lost. This might be unacceptable in certain applications. Hence, MP/AOS provides the Flush option on ?WRITE to ensure that the state of the file is kept up to date.

When a ?WRITE system call with the Flush option returns, the user is assured that all file system data associated with that I/O channel is written to disk before the return to the user program is taken. This helps ensure that the file information on disk is constantly being updated, enabling the user to create checkpoint (environment) records, and ensuring valid restart points in case of system crash.

The Flush option on ?READ affects character devices only, causing any characters currently held in the system buffer to be discarded.

The buffering mechanism is bypassed altogether when data transfers requested are *block-aligned*, that is, when:

- the file pointer is set to a multiple of 512 bytes before the transfer,
- the transfer request is for a multiple of 512 bytes,
- the destination address in memory is word-aligned.

Nonpended I/O

MP/AOS offers the user the option of issuing nonpended I/O calls. When a nonpended system call is issued, the system does not suspend the calling task until the completion of the call; instead, it creates (transparently to the user) a system task to execute the nonpended call, while the original task is free to continue its operation.

The user program can issue a system call to determine when any one of its nonpended calls is complete, and to receive the task ID of the completed call. This permits the program to obtain information about the completed call's outputs and minimizes delays in ascertaining task status.

I/O Techniques

MP/AOS provides a number of different techniques for I/O transfers. These techniques are also valid for data transfers to and from user-defined memory segments, regardless of whether these segments are mapped to the user's logical address space.

Dynamic I/O allows a specified number of bytes to be read and written. The efficiency of this mode of transfer may be improved by moving entire 512 byte disk blocks.

Data Sensitive I/O transfers bytes up to a maximum number or until a predefined delimiter is encountered. Default delimiters are provided by MP/AOS, but user-defined delimiter tables are supported up to a maximum number specified by the user when generating the system.

Extended (Direct Segment) I/O can be used in either dynamic or data-sensitive mode; that is, bytes can be transferred by count or until a predefined delimiter is encountered. Normally, data transfers take place between the device and the user's logical address space. The extended I/O facility permits data to be transferred directly between any memory segment and a disk device. If the memory segment is user-defined, it need not be mapped to user address space. Extended I/O is valid only for block devices such as disks.

MP/AOS I/O devices are divided into those with *directory* structures such as mounted disks and those without directory structures—*character devices* such as consoles and line printers.

Disks

User calls mount and dismount disk devices to permit the introduction of new media to the system in an orderly fashion. The system performs consistency checks when a disk is mounted and dismounted, notifying the user if the disk needs repair. The user can then invoke the disk FIXUP program to repair the disk. If a problem is encountered when the system master disk is booted at system start, the FIXUP program is run automatically. The user can ascertain disk status information for mounted disks.

Opening a channel to a disk without mounting the disk causes the disk to be accessed as a single file with an element size equal to the number of blocks on the disk.

Disks that have not been software formatted for MP/AOS or MP/OS must be prepared before they can be mounted. This is done by means of the disk initializer utility, DINIT.

Magnetic Tape

MP/AOS supports magnetic tape devices as part of its library. Magnetic tape devices are also supported by the MOVE utility. For direct access to magnetic tape controllers without use of the MOVE utility, the user program must be bound with the tape routine library. This will introduce into the program the service routines identifying the magnetic tape controller as a special user device.

Character Devices: Terminals

Terminals are logically handled as two different devices: the keyboard for input and the printer or CRT for output.

MP/AOS provides a sophisticated console interface compatible with MP/OS. Console *characteristics* (the attributes controlling the receiving and transmission of data) are user modifiable with system calls.

Console characteristics afford numerous facilities without programming intervention, for example, echoing characters received from the keyboard, echoing Form Feeds as ↑L to prevent them from erasing the CRT screen, executing Rubouts, and converting to uppercase on output.

The terminal's hardware characteristics such as the baud rate are user-specified on system generation. Hardware with programmable characteristics is user-modifiable under software control.

The system assigns special functions to certain control characters and to its two-character control sequences. For example, control characters serve as standard delimiters for data-sensitive data transfers, while control sequences provide the ability to pass an interrupt to a program, to terminate a currently executing program, and to save the program state in a break file or to return control to the debugger.

Character Devices: Line Printers

The system handles line printers similarly to console output devices. Line printer characteristics are a subset of console characteristics.

MP/AOS supports data channel line printers as part of its library.

Line printer support includes the SPOOLER utility, spooling through Interprocess Communication, and direct access to printers through an assembly language or Pascal interface.

For direct access to data channel line printers without the SPOOLER utility, the user program must be bound with the data channel routine library that makes the appropriate service routines available.

MP/AOS input and output system calls are summarized in Table 2.5.

Call	Function	Option
?AWAIT	Await completion of nonpended system call	AY (return when any nonpended call is completed) CK (check call completion; error return if call incomplete)
?CLOSE	Close an I/O channel	DE (delete the file)
?DISMOUNT	Remove a disk from the system	
?DSTAT	Get disk status information	
?GPOS	Get the file position	
?MOUNT	Introduce a disk to the system	
?OPEN	Open an I/O channel	AP (files: open for append; character devices: suppress Form Feeds) CR (create file) DE (delete existing file) EX (exclusive access) NZ (don't zero blocks on allocation) UC (unconditionally create file)
?READ	Read data from a device or file	DS (data-sensitive read) FL (character devices: flush buffer before reading) IX (only with DS: ignore input after maximum byte count or delimiter) NP (nonpended call) PKT (block devices: extended I/O
?RESET	Close multiple I/O channels	
?SCHS	Set channel specifications	
?SPOS	Set current file position	EF (cause error return on end-of-file)
?WRITE	Write data to a device or file	DS (data-sensitive write) EF (cause end-of-file error return on attempt to extend file) FL (flush current block to disk) NP (nonpended call) PKT (block devices: extended write)

Table 2.5 Input/output system calls summary

User Device Support

MP/AOS capabilities permit user control of I/O protection for system and user devices, user-written device service routines, and user manipulation of data channel and burst multiplexor channel (BMC) map slots. These facilities make it possible for programmers to perform input and output with custom devices, to take advantage of their device's interrupt facility, and to perform data transfers through data channel and BMC control. The maximum number of user-written device drivers is a system generation parameter.

Device service routines for user devices reside in user process space. A system call specifying an interrupt handler definition packet introduces custom devices and their interrupt service routines to the system. MP/AOS also provides support for user devices connected to a single line of Asynchronous Line Multiplexor (ALM) or Asynchronous Synchronous Line Multiplexor (ASLM) boards. The interrupt

handler definition packet allows the system to locate the device's interrupt service routine once an interrupt request is detected. The ECLIPSE architecture has the capability of implementing up to sixteen (16) levels of priority interrupts.

Data Channel and BMC Map Manipulation

The data channel facility makes possible data transfers between memory and registers in the device controller. Depending on the device's design, the user can select either data channel (DCH) or burst multiplexor channel (BMC) I/O for a user-defined device.

Data channel and BMC transfers are performed across *data channel maps* and *BMC maps* (translation units for the DCH and the BMC) in units whose size is device specific. All data channel I/O for Data General devices is premapped by MP/AOS in conformance with the data channel capabilities of each device.

To enable data transfer through data channel or BMC control with user-built devices, MP/AOS provides system calls to manipulate a portion of the four data channel maps as well as the BMC map. Data channel and BMC maps can be set up at either the task or interrupt level.

Data Channel and BMC Map Organization

The four data channel maps are lettered A to D. Each of these maps contains 32 *slots,* or words. The BMC map contains 1024 slots.

Slot Number	Map
0-31	A
32-63	B
64-95	C
96-127	D
128-1151	BMC

Table 2.6 Data channel and BMC map slot numbering

The slots are numbered consecutively starting from 0 for the first slot in map A, through 1151, the last slot in the BMC map. Table 2.6 summarizes this scheme.

Each data channel map slot word corresponds to a 1-Kword range of logical data channel addresses, from 0 through 1023 decimal. These addresses are also numbered consecutively within each map, from 0 for the first address in the first slot of each map, through 32767_{10} for the last address in the thirty-second slot. Figure 2.7 illustrates this scheme.

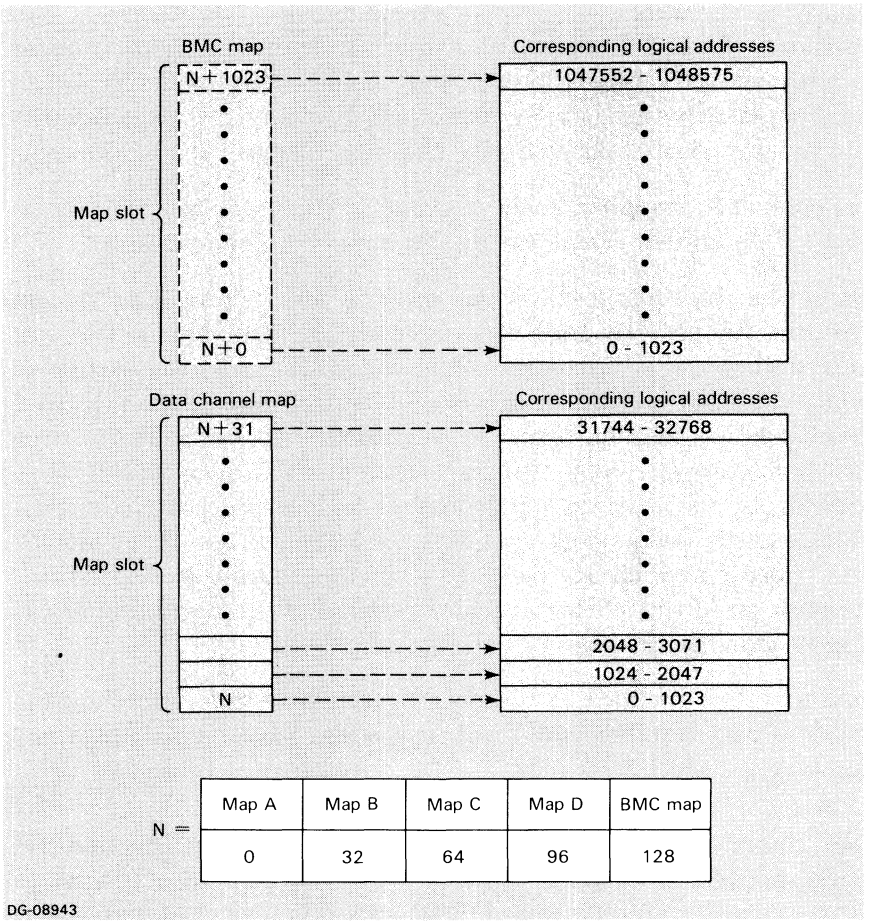

Figure 2.7 Data channel and BMC map slots and their range of corresponding logical addresses

In this fashion, the data channel map slot numbers serve to identify the map selected, as well as to reference the logical data channel address passed by the user program to the device with the I/O instruction. For example, slot 60 references map B and a logical data channel address range of 28672_{10} to 29696_{10}.

Data Channel and BMC Mapping Procedures

The following sequence of operations sets up data channel and BMC maps:

1. Allocate data channel (or BMC) map slots; depending on the device's capabilities, allocations spanning several maps are possible.
2. Translate user logical address for the start of transfer into a physical page number.
3. Set up data channel map: store user physical page number in the appropriate data channel or BMC slot, where it serves as a pointer to a buffer in the user address area.
4. Initiate transfer — enable I/O.

Following the allocation of data channel map slots, a system call translates the user's logical page number from which to transfer data in or out into a physical page number in memory. When mapping is requested, the system stores that physical page number into the previously allocated data channel slot specified by the user.

Although the mapping itself is done by the system, the user is responsible for computing the proper data channel addresses.

When the user program issues an I/O instruction, the system identifies the device and loads the *starting data channel logical address* into the accumulator. This address permits the system to identify the slot containing the user's physical page number, as well as the relative position from the beginning of that page for starting the data transfer. (For transfers which are aligned with page boundaries, the starting data channel logical address is the first address in the range corresponding to the particular map slot; for data transfers which are not page-aligned, a word offset specifying the position of the start of transfer relative to the beginning of the page is added to the beginning range address.)

Data are then mapped to the correct user area referenced by the physical page number contained in the data channel or BMC map slot.

Controlling Access to All I/O Devices

User interrupt handling routines control I/O protection and are therefore able to issue I/O instructions anywhere in a process at either the task or interrupt level. I/O protection is controlled by I/O enable and disable calls.

Under MP/AOS the enable/disable calls simultaneously control the I/O / LEF mode and the I/O protection bit. That is, enabling I/O disables both LEF mode and the I/O protection bit at once; disabling I/O enables LEF mode as well as the I/O protection bit. The system's interrupt service routine always enables I/O instructions before transferring control to the device's interrupt service routine.

Table 2.7 summarizes the MP/AOS system calls supporting custom devices.

Call	Function	Option
?ALMP	Allocate data channel map slots	MA (data channel map A) MB (data channel map B) MC (data channel map C) MD (data channel map D) BMC (burst multiplexor channel map)
?DEMP	Deallocate data channel map slots	
?DSBL	Disable I/O instructions/enable LEF mode	
?ENBL	Enable I/O instructions/disable LEF mode	
?GMRP	Get physical page number	
?IDEF	Define an interrupt handling routine	
?IPEND	Pend awaiting interrupt activity	
?IRMV	Remove an interrupt handling routine	
?IUNPEND	Unpend a task from interrupt handling routine	BD (unpend all tasks) ER (unpended tasks take error return from ?PEND, ?IPEND) ID (unpend on task identifier, not event number)
?IXIT	Exit from an interrupt handling routine	
?LDEF	Define a line interrupt handling routine	
?LRMV	Remove a line interrupt handling routine	
?LXIT	Exit from a line interrupt handling routine	
?STMP	Set up data channel map	

Table 2.7 User device support system calls summary

Process Debugging & Histogramming

Process Debugging

MP/AOS allows users to debug a program without having to bind a debugger with it. FLIT, the MP/AOS process debugger, is an independent program that runs at the same time as the executing program but within a different process space. Thus, the debugger program and the executing program are independent processes running in parallel, and residing in main memory during debugging.

The debugger process is activated whenever the system blocks the executing program. Program blocking occurs any time MP/AOS detects user-specified or automatic breakpoints (for instance, a console ↑C↑D sequence) while screening the executing program. The system then generates a *signal* that is intercepted by the debugger; MP/AOS blocks the program, and the debugger process takes control. A brief description of the FLIT debugger is found in Chapter 3.

MP/AOS support for users desiring to write their own debugger programs includes several system calls as summarized in Table 2.8: an Await Signal call to intercept signals from the executing program, turn over control to the debugger, and resume program execution; and system calls allowing the debugger program to obtain all task identifiers, and to examine and modify task states and memory locations in the executing program.

Call	Function	Option
?GIDS	Get task identifiers	
?RDMEM	Read memory	
?RDST	Read task state	
?WRMEM	Write to memory	
?WRST	Modify task state	
?WSIG	Await signal/resume execution	RE (resume execution)

Table 2.8 Debugger system calls summary

Process Histogramming

MP/AOS provides two system calls and a utility program enabling the user to monitor program execution and determine bottlenecks. The PROFILE utility program (summarized in Chapter 3) is built around the system calls and provides an easy-to-use interface and an easy-to-read report. The system calls can serve for user-written execution monitoring programs.

The two MP/AOS program monitoring system calls (see Table 2.9) allow continual sampling of programs executing in process space to determine a statistical profile of the execution time spent in each region of the address space. The frequency distribution generated by these system calls is referred to as a *histogram*. When generating an MP/AOS system, the user specifies the number of concurrent histogram operations that can be carried out on executing programs system-wide.

Program monitoring data is gathered and placed in temporary storage locations by the Enable Histogram call; the Disable Histogram call causes the data gathering to stop and makes a summary of the data accessible to the user.

User-selectable histogram parameters include range of addresses, monitoring frequency per second, and the interval size used to create the histogram.

Call	Function
?DHIS	Disable histogramming
?EHIS	Histogram a process

Table 2.9 Histogrammer system calls summary

Cross Development on AOS

The MP System Call Translator software package supplied with MP/AOS may aid in the development of MP/AOS programs on ECLIPSE line computers under the Advanced Operating System (AOS). The System Call Translator translates a subset of MP/AOS calls into their AOS counterparts. Programs developed under MP/AOS using this subset of calls can thus be transported to AOS by rebinding them with the System Call Translator object module and subroutine library file.

Some programs developed under AOS using the System Call Translator can be moved to MP/AOS with no modification except for rebinding. The MP/AOS Macroassembler and Binder are two such programs: using the System Call Translator under AOS, they require only rebinding to be moved to MP/AOS or to MP/OS. This allows the same sources and object modules to be used over the entire ECLIPSE computer line.

Library Routines

MP/AOS provides a number of convenient functions implemented as library routines rather than as system calls in the interest of saving system space and improving speed of execution.

Library routines are called in the same way as system calls; however, the code implementing the function is part of the user address space, rather than of system memory.

MP/AOS library routines perform such functions as overlay loading and release, suspending the operation of a program for a specified time period, providing several timing facilities important for real-time operations support, setting the searchlist, and reading a message from an MP/AOS error message file. Table 2.10 summarizes currently available MP/AOS library routines.

Routine	Function
?CDAY	Convert system time/date to date
?CTOD	Convert system time/date to time of day
?DELAY	Delay execution of a task
?ERMSG	Retrieve a system error message
?FDAY	Convert date
?FTOD	Convert time of day
?GDAY	Get the current date
?GNFN	Get next filename in working directory
?GTOD	Get the current time of day
?MSEC	Convert a time to milliseconds
?OVLOD	Load an overlay
?OVREL	Release an overlay
?SDAY	Set the system calendar
?SLIST	Set the searchlist
?STOD	Set the system clock
?TMSG	Translate a CLI-format message

Table 2.10 Library routines summary

Table 2.11 summarizes miscellaneous MP/AOS system calls.

Call	Function
?EQT	Set up system call
?GTIME	Get current system time and date
?GTMSG	Get an interprogram message
?STIME	Set current system time and date

Table 2.11 Miscellaneous system calls

MP/AOS Utilities

This chapter summarizes the main features of the MP/AOS utilities. We have classified these utilities by function into three major groups: program development utilities, file management utilities, and system management utilities. The listing of utilities in each group includes references to the manual in which each utility is documented.

Program Development Utilities

Table 3.1 lists the MP/AOS utilities which can be used for program development, as well as the manual in which each utility or group of utilities is documented.

Utility	Documentation
Command Line Interpreter (CLI)	*MP/AOS Command Line Interpreter* (DGC No. 069-400201)
Speed Text Editor (SPEED)	*MP/AOS SPEED Text Editor* (DGC No. 069-400202)
Slate Text Editor (SLATE)	*MP/AOS SLATE Text Editor* (DGC No. 069-400209)
Macroassembler (MASM) Binder (BIND) Library Editor (LED) Symbol Cross Reference Analyzer (SCAN)	*MP/AOS Macroassembler, Binder, and Library Utilities* (DGC No. 069-400210)
Histogrammer (PROFILE) Process Debugger (FLIT) Process Monitor (OPM)	*MP/AOS Debugger and Performance Monitoring Utilities* (DGC No. 069-400205)
Text Control System (TCS) Build Program (BUILD) Find Program (FIND)	*MP/AOS Advanced Program Development Utilities* (DGC No. 069-400208)

Table 3.1 Program development utilities

Command Line Interpreter (CLI)

The Command Line Interpreter (CLI), the primary user interface to the operating system, provides macro and *textual substitution* facilities in addition to built-in commands and pseudo-macros, as a means of manipulating the system and user environments. CLI commands can be abbreviated to the shortest character strings that uniquely identify the commands. On-line information about CLI commands and topics is displayed at the terminal via the HELP facility.

For most program development configurations, the CLI begins running as soon as the system is initialized. CLI commands allow the user to control peripheral devices, manage files and processes, facilitate interprocess communication, and execute programs. Any CLI command or macro may be used as a textual substitution whose value serves as an argument to another command or macro.

Speed Text Editor (SPEED)

Speed is a character-oriented text editor that also provides file management and CLI execution capabilities. Command sequences and macros may be saved in buffers or files allowing for automatic re-execution.

Speed editing commands insert, delete, and change individual characters, lines, or blocks of text, and search for specific words, strings, or patterns. The display mode option provides a constant image of most recent updates on a CRT screen.

Thirty-six buffers provide workspaces for creating, modifying, and merging text files. File operations are not restricted to the current working directory.

While in Speed, users can execute CLI commands and any other MP/AOS programs and utilities.

Slate

SLATE is a screen-oriented text editor using screen edit functionality, multiple text buffers, macro programming, and CLI execution capabilities. Command sequences and macro definitions can be saved in files, allowing for automatic re-execution.

SLATE features three major operating modes: Page Edit mode, Key Macro Definition mode, and Command mode. Command sequences and macro key definitions can be written in either Page Edit or Command mode. SLATE allows up to 25 key definitions at any one time.

Ten logical text buffers provide workspaces for creating, modifying, and merging text files. File operations are not restricted to the current working directory.

SLATE editing commands insert, delete, and modify individual characters, words, lines, or blocks of text and search for specific words, strings, or patterns. With keystroke commands, restoration of deleted text, text reformatting, alphabetic case conversion and a variety of search modes are also supported. The order of command execution can be modified by means of pseudo-commands. A HELP command displays information about SLATE commands and topics.

While in SLATE, users can execute CLI commands and other MP/AOS programs and utilities.

Macroassembler (MASM)

The macroassembler produces a single object module file from one or more user-specified assembly language source files. In addition to translating symbolic assembly language source code, the macroassembler expands any macro calls in the source program to their full, predefined macro definitions.

User-selectable output from the macroassembler includes:

- the object module file,
- a source language program listing (with any errors indicated),
- a cross-reference list of all program symbols,
- an error summary list with the numbers of all source lines containing errors.

An object module produced by the macroassembler requires further processing by the MP/AOS binder before it becomes an executable program.

A special compatibility version of the macroassembler's permanent symbol table file allows assembly of files developed under AOS, or MP/OS, and guarantees bind-time portability to either of these operating systems.

Binder (BIND)

The binder creates an executable program file by combining user-specified object modules and library files with required system information. When potential run-time errors are detected, the binder does not produce a program file.

In creating an executable program (.PR) file from object modules (.OB) files, the binder resolves run-time memory space considerations for relocatable object modules, manages program overlays, and sets up shared common areas.

User-selectable output from the binder includes:

- an executable program module,
- a symbol table file,
- a listing of any binder error messages,
- symbol/location ordered listings,
- a comprehensive space allocation summary (that is, a display of the actual modules used from a library, together with information about the memory use of each module).

Arguments to the CLI command, XEQ BIND, specify the object modules to be bound together as a program and name any libraries to be inspected for modules containing unresolved symbols.

Programs developed under AOS using the compatibility package may be run under MP/AOS after rebinding. MP/AOS is bind-time compatible with all MP/OS features except for user device interrupt handlers, which need reassembling with minor modifications.

The Library Editor (LED)

The Library Editor (LED) creates an object module library file consisting of one or more object modules or libraries. The individual library modules may be routines supplied with the system (such as the mathematical subroutines) or user-written modules already processed by one of the MP/AOS language compilers or the macroassembler.

LED also allows for the insertion, replacement, and deletion of object modules in existing library files.

Symbolic Cross-Reference Analyzer (SCAN)

SCAN processes a library or a group of object modules and produces a cross-reference listing of entry definitions and external uses for all entry and external symbols.

The SCAN program detects multiply-defined, undefined, and defined but unused global symbols over a range of object modules and libraries. Programmers can use SCAN to determine whether a symbol name has already been used, and to check that modules to be bound together do not have conflicting symbol definitions.

Histogrammer (PROFILE)

The PROFILE program constructs a probabilistic execution profile of a specified program. The profile (sometimes called a histogram) is useful in identifying program bottlenecks.

To construct the histogram, PROFILE uses the ?HIST call, which inspects the CPU's program counter at regular, selectable intervals and notes whether it is being used by

the system idle loop,
a process other than the one being profiled,
user-written code of the program being profiled,
system calls within the program being profiled.

If, at the time of the inspection, the CPU is being used by the profiled program, then the address of the instruction being executed is noted.

PROFILE produces a tabularized report of CPU usage, and of the times spent in each region of the profiled program. Using this report, programmers can see which areas of their programs are most executed and which are therefore the best candidates for optimization.

Process Debugger (FLIT)

The process debugger is an interactive system utility providing access to a task's memory locations, registers, and status information. Programs to be debugged with FLIT need no special handling at coding, assembly, or bind time, since the debugger runs as a separate process. This also means that no user address space is taken up.

In response to appropriate signals generated by the system, FLIT receives control of the programs to be debugged. Once FLIT has control, the program and its run-time information may be interactively examined or modified. With FLIT, users may:

- examine task status,
- modify task control information,
- set unconditional and conditional breakpoints,
- alter display and addressing modes,
- define macros and script files,
- use a single keystroke for tracing and returning from pointer chains.

Process Monitor (OPM)

The process monitor provides a dynamic tabular display of current system resource allocation and status. The information displayed is user-selectable, and may include information about the status, priority, CPU usage, and I/O transfer rates of every process.

Text Control System (TCS)

TCS is a method for managing files that are derived from or are versions of a single file. Under TCS, all the information necessary for the reconstruction of any version of the file is kept in a single master file. Access to the master file is controlled and monitored by TCS so that only one person at a time can update a file. This system decreases disk storage requirements by reducing the amount of redundantly stored information. TCS can tell the programmer what was changed, when, by whom, and for what reason.

The Build Utility (BUILD)

The BUILD utility facilitates the process of creating a product through transformations and combinations of other files. For example, BUILD addresses the problems encountered during the creation of a program file through the editing, assembling, compiling, and binding of other files.

BUILD is used whenever a new version of a product is constructed by a repetition of the procedure used to construct the original version. BUILD maintains a record of the original procedure and of the status of each file used in the process. Using this information, BUILD determines the minimal amount of work required to produce the intended product, bypassing file modifications not needed for the new version.

BUILD eliminates the common mistake of creating a version of a product, but forgetting to compile one of the constituent modules.

BUILD is fully integrated with TCS, both for the selection of input files and for setting the revision levels of the files supplied to BUILD and those files re-created by BUILD.

The Find Utility (FIND)

The FIND utility detects and identifies occurrences of patterns in text files. It can be used in conjunction with or instead of text editors, and is especially useful for work on modems with low baud rates.

File Management Utilities

The file management utilities, documented in *MP/AOS File Utilities* (DGC No. 069-400204) are:

File Editor (FEDIT)
File Display and Comparison Utility (FDISP)
Source Compare Program (SCMP)
Move Utility (MOVE)
AOS Transfer Utility (AOSMIC)
File Transfer Utility (FOXFIRE)

File Editor (FEDIT)

The file editor program allows the user to examine and modify the files of an MP/AOS system. While FEDIT may be executed on any file, it is typically most useful for program files and data files.

Addresses as well as data can be displayed in different formats including numeric with variable radix, ASCII, and string formats. Automatic modification of files is possible by redirecting FEDIT input from a terminal to a disk file. Script output files may also be generated for later use as script input files.

File Display and Comparison Utility (FDISP)

The file display and comparison utility displays a file's contents in readable form. When used to compare the contents of two files, FDISP lists only those parts of the files which are at variance.

Addresses and data can be displayed in numeric format with variable radix, byte-by-byte format, and ASCII. The user can also specify a comparison of entire files or of specific portions in each file, as well as the width of the output display.

FDISP facilitates analysis and comparison of program and other nontext files.

Source Compare Program (SCMP)

The source compare program compares two source programs line by line. The output enumerates the changes necessary in the first file so that it may correspond to the second file.

Command switches allow the user to redirect the output, to change the default resynchronization of lines, and to select case sensitivity for the input.

MOVE Utility (MOVE)

The MOVE utility allows the transfer of file copies from one directory to another, optionally specified by template. Files may also be moved between disk and magnetic tape devices by means of the appropriate command switches. In addition, MOVE performs a multi-volume dump to diskette; that is, it permits the transfer of any set of directories and files onto more than one diskette. The tape format of MOVE is compatible with AOS. Other switches allow reversal of the transfer direction, automatic selection of files to be moved in accordance with date of their modification by the user, and nondeletion of existing files in the destination directory.

AOS File Transfer Utility (AOSMIC)

The AOS file transfer utility enables the user to manipulate MP/AOS and MP/OS disks and files on an AOS system. By running this utility while under AOS:

- MP/AOS files may be moved between an MP/AOS disk and the AOS system;
- MP/AOS directories may be created;
- MP/AOS files may be renamed and deleted;
- the contents of the MP/AOS disk may be listed.

File Transfer Utility (FOXFIRE)

The file transfer utility enables the user to transfer disk files between any combination of MP/OS, MP/AOS, and AOS systems over asynchronous communication lines.

System Management Utilities

The following utilities can be used for system management:

System Generation (SYSGEN)
Disk Initializer (DINIT)
Disk Repair (FIXUP)
Spooler Utility (SPOOLER)
Error Logger (ELOG)

For further reference see *MP/AOS System Generation and Related Utilities* (DGC No. 016-400206). See also *Loading MP/AOS* (DGC No. 093-400051).

System Generation (SYSGEN)

SYSGEN is an interactive utility used to generate new MP/AOS systems tailored to specific user needs. It prompts the user for input and executes the MP/AOS assembler and binder to build a suitable system file. The results of the dialogue may be stored in a *script file* for input to a later SYSGEN session. In the absence of user-specified values, default values are supplied.

SYSGEN creates program development systems: general-purpose, disk-based systems that provide all MP/AOS features.

User-selectable input includes processor type, disk controller type, number of devices, the characteristics of terminals and line printers, and MP/AOS system parameters. SYSGEN enables users to generate MP/AOS systems that are optimized for their particular application environment.

Disk Initializer (DINIT)

The disk initializer utility performs two functions: it software formats disks for the storage of MP/AOS data structures; it also installs the files necessary to make the disk a system disk. The user may select either or both of these functions.

Since MP/AOS disk format is identical with MP/OS, disks formatted for MP/OS with DINIT can be mounted on an MP/AOS system and file management operations can be performed on them. To create MP/AOS disks under AOS, MDINIT is used to format the disk.

DINIT is an interactive utility, but optional command switches allow the user to specify information on the command line, thereby eliminating part or all of the interactive dialogue.

Disk Repair (FIXUP)

The FIXUP utility repairs defective disk structures. It exists in two versions: one version is installed on the disk by the Disk Initializer program (DINIT) and runs automatically without user intervention.

The second version is an interactive program which may be invoked by the user. FIXUP may be run on disks developed under either MP/OS or MP/AOS. For MP/AOS disks developed under AOS, a similar utility, MFIXUP, may be invoked.

Spooler Utility (SPOOLER)

The SPOOLER controls the queueing of files to each line printer on the MP/AOS system. The SPOOLER accepts print queue entries from CLI commands and the Interprocess Communication (IPC) facility; maintains the queue on a first-in, first-out basis; and activates the printer when there are queue entries.

CLI commands allow the user to queue, display, hold, release, and cancel SPOOLER entries. Options to the queue command permit wrapping of long lines, inclusion of a banner, printing of multiple copies, and requesting of notification when printing is completed.

The SPOOLER queues as many as 64 requests at a time, prints ASCII text up to 132 characters per line, and supports data channel printers.

Printing without the SPOOLER utility is also possible through assembly language or Pascal interfaces to both programmed I/O and data channel printers (the latter accessed via a library of subroutines).

Error Logger (ELOG)

The error logger prepares a report that may be written to a console, line printer, or disk file about system hardware performance. The information used to generate the report is gathered by the system logger, which is optionally initialized at system generation. For every memory component and I/O device, the report lists the date, time, and nature of every error detected by the system. This report is useful to system managers and field engineers for overall system monitoring, and for preliminary diagnostics.

High-Level Languages Under MP/AOS

This chapter summarizes the main features of the high-level languages running under MP/AOS. These languages are MP/Pascal, SP/Pascal, MP/FORTRAN IV, and MP/BASIC.

MP/Pascal

MP/Pascal, an extended, structured programming language based on Pascal, produces understandable and maintainable programs requiring less execution space than other high-level languages. MP/Pascal's extensions to Pascal include:

- separate compilation modules for sharable routines and data,
- assembly language interface,
- dedicated application prommability,
- dynamic string data types for text manipulation,
- multitasking,
- MP/OS and AOS compatibility as a cross-development aid,
- full access to MP/AOS system functions.

User-selectable output from the MP/Pascal compiler includes a source language program listing (with any errors indicated) and, when no compilation errors exist, an object module file; the object module produced requires further processing by the MP/AOS binder before it becomes part of an executable program.

SP/Pascal

SP/Pascal, an extended system-development Pascal, has all of the features of MP/Pascal. Additional system implementation extensions equip SP/Pascal with the following unique features:

- generates highly efficient machine code for the ECLIPSE architecture,
- introduces exception handling and routine signaling support,
- utilizes floating-point hardware for single- and double-precision real arithmetic operations,
- provides bit-level packed data storage,
- uses hardware character instruction set option for string manipulation,
- allows structured constants for use as constants or literals,
- permits scalar expressions in most constant contexts,
- provides WHOLE type with a 0..65535 subrange for unsigned full-word operations,
- introduces extensions to the WITH statement for easier record copying,
- is available under the AOS operating system with the system call emulator.

User-selectable output from the SP/Pascal compiler, as with MP/Pascal, includes a source language program listing (with any errors indicated) and, when no compilation errors exist, an object module file for further processing by the MP/AOS binder. For debugging purposes, the compiler can generate a disassembler report which

lists the ECLIPSE instruction equivalent of the SP/Pascal source program.

MP/FORTRAN IV

MP/FORTRAN IV, an extended, ANSI 1966 standard programming language, produces comprehensive and maintainable user programs. Formatted I/O gives FORTRAN output capabilities generally not available in higher-level languages. Efficiency is enhanced by the optional optimization capability of the MP/FORTRAN IV compiler and the ECLIPSE floating-point hardware that speeds up FORTRAN floating-point operations.

An interface with assembly language provides for critical paths and user device handling; compatibility of language conventions makes MP/FORTRAN IV programs transportable across all Data General hardware and operating systems.

MP/FORTRAN IV extensions to original ANSI 1966 standard FORTRAN IV include:

- full access to system multitasking capabilities;
- dedicated application prommability;
- comprehensive file I/O facilities that interface to system file I/O and file management commands;
- extensive built-in library functions;
- run-time format specification, task scheduling, and coordination at runtime;
- ability to utilize hardware stack for individual tasks;
- double-precision real, complex, and integer arithmetic;
- quoted string constants as well as Hollerith constants;
- full MP/AOS system support for swapping, chaining, and overlaying portions of large programs.

The MP/FORTRAN compiler produces a single assembler source file that requires further processing by the MP/AOS assembler and binder before becoming an executable program. The optional optimizer program can add run-time efficiency. User-selectable compiler output includes a source language program listing (with any errors indicated) and, when no compilation errors exist, the object file.

MP/BASIC

MP/BASIC, an extended, ANSI-standard programming language, produces understandable and maintainable user programs. BASIC is interactive, and easy to learn and use. Among its major features are a real data type and string manipulation capabilities.

MP/BASIC's extensions to ANSI standard BASIC include integer data type, string dimensioning and concatenation, substrings, letter-digit array names, extensive built-in string and mathematical

functions, and fixed and variable length file manipulation. Program swapping and chaining and exception handling are additional features. An interface with assembly language provides for user device handling.

Index

DG OFFICES

NORTH AMERICAN OFFICES

Alabama: Birmingham
Arizona: Phoenix, Tucson
Arkansas: Little Rock
California: Anaheim, El Segundo, Fresno, Los Angeles, Oakland, Palo Alto, Riverside, Sacramento, San Diego, San Francisco, Santa Barbara, Sunnyvale, Van Nuys
Colorado: Colorado Springs, Denver
Connecticut: North Branford, Norwalk
Florida: Ft. Lauderdale, Orlando, Tampa
Georgia: Norcross
Idaho: Boise
Iowa: Bettendorf, Des Moines
Illinois: Arlington Heights, Champaign, Chicago, Peoria, Rockford
Indiana: Indianapolis
Kentucky: Louisville
Louisiana: Baton Rouge, Metairie
Maine: Portland, Westbrook
Maryland: Baltimore
Massachusetts: Cambridge, Framingham, Southboro, Waltham, Wellesley, Westboro, West Springfield, Worcester
Michigan: Grand Rapids, Southfield
Minnesota: Richfield
Missouri: Creve Coeur, Kansas City
Mississippi: Jackson
Montana: Billings
Nebraska: Omaha
Nevada: Reno
New Hampshire: Bedford, Portsmouth
New Jersey: Cherry Hill, Somerset, Wayne
New Mexico: Albuquerque
New York: Buffalo, Lake Success, Latham, Liverpool, Melville, New York City, Rochester, White Plains
North Carolina: Charlotte, Greensboro, Greenville, Raleigh, Research Triangle Park
Ohio: Brooklyn Heights, Cincinnati, Columbus, Dayton
Oklahoma: Oklahoma City, Tulsa
Oregon: Lake Oswego
Pennsylvania: Blue Bell, Lancaster, Philadelphia, Pittsburgh
Rhode Island: Providence
South Carolina: Columbia
Tennessee: Knoxville, Memphis, Nashville
Texas: Austin, Dallas, El Paso, Ft. Worth, Houston, San Antonio
Utah: Salt Lake City
Virginia: McLean, Norfolk, Richmond, Salem
Washington: Bellevue, Richland, Spokane
West Virginia: Charleston
Wisconsin: Brookfield, Grand Chute, Madison

DG-04976

INTERNATIONAL OFFICES

Argentina: Buenos Aires
Australia: Adelaide, Brisbane, Hobart, Melbourne, Newcastle, Perth, Sydney
Austria: Vienna
Belgium: Brussels
Bolivia: La Paz
Brazil: Sao Paulo
Canada: Calgary, Edmonton, Montreal, Ottawa, Quebec, Toronto, Vancouver, Winnipeg
Chile: Santiago
Columbia: Bogata
Costa Rica: San Jose
Denmark: Copenhagen
Ecuador: Quito
Egypt: Cairo
Finland: Helsinki
France: Le Plessis-Robinson, Lille, Lyon, Nantes, Paris, Saint Denis, Strasbourg
Guatemala: Guatemala City
Hong Kong
India: Bombay
Indonesia: Jakarta, Pusat
Ireland: Dublin
Israel: Tel Aviv
Italy: Bologna, Florence, Milan, Padua, Rome, Tourin
Japan: Fukuoka, Hiroshima, Nagoya, Osaka, Tokyo, Tsukuba
Jordan: Amman
Korea: Seoul
Kuwait: Kuwait
Lebanon: Beirut
Malaysia: Kuala Lumpur
Mexico: Mexico City, Monterrey
Morocco: Casablanca
The Netherlands: Amsterdam, Rijswijk
New Zealand: Auckland, Wellington
Nicaragua: Managua
Nigeria: Ibadan, Lagos
Norway: Oslo
Paraguay: Asuncion
Peru: Lima
Philippine Islands: Manila
Portugal: Lisbon
Puerto Rico: Hato Rey
Saudi Arabia: Jeddah, Riyadh
Singapore
South Africa: Cape Town, Durban, Johannesburg, Pretoria
Spain: Barcelona, Bibao, Madrid
Sweden: Gothenburg, Malmo, Stockholm
Switzerland: Lausanne, Zurich
Taiwan: Taipei
Thailand: Bangkok
Turkey: Ankara
United Kingdom: Birmingham, Bristol, Glasgow, Hounslow, London, Manchester
Uruguay: Montevideo
USSR: Espoo
Venezuela: Maracaibo
West Germany: Dusseldorf, Frankfurt, Hamburg, Hannover, Munich, Nuremburg, Stuttgart

DataGeneral

Ordering Technical Publications

TIPS is the Technical Information and Publications Service--a new support system for DGC customers that makes ordering technical manuals simple and fast. Simple, because TIPS is a central supplier of literature about DGC products. And fast, because TIPS specializes in handling publications.

TIPS was designed by DG's Educational Services people to follow through on your order as soon as it's received. To offer discounts on bulk orders. To let you choose the method of shipment you prefer. And to deliver within a schedule you can live with.

How to Get in Touch with TIPS

Contact your local DGC education center for brochures, prices, and order forms. Or get in touch with a TIPS administrator directly by calling (617) 366-8911, extension 4086, or writing to

Data General Corporation
Attn: Educational Services, TIPS Administrator
MS F019
4400 Computer Drive
Westborough, MA 01580

TIPS. For the technical manuals you need, when you need them.

DGC Education Centers

Boston Education Center
Route 9
Southboro, Massachusetts 01772
(617) 485-7270

Washington, D.C. Education Center
7927 Jones Branch Drive, Suite 200
McLean, Virginia 22102
(703) 827-9666

Atlanta Education Center
6855 Jimmy Carter Boulevard, Suite 1790
Norcross, Georgia 30071
(404) 448-9224

Los Angeles Education Center
5250 West Century Boulevard
Los Angeles, California 90045
(213) 670-4011

Chicago Education Center
703 West Algonquin Road
Arlington Heights, Illinois 60005
(312) 364-3045

DataGeneral

Technical Products Publications

Comment Form

Please help us improve our future publications by answering the questions below. Use the space provided for your comments.

Title: _____

Document No. 069-400200-00

Yes	No		
☐	☐	Is this manual easy to read?	○ You (can, cannot) find things easily. ○ Other: ○ Language (is, is not) appropriate. ○ Technical terms (are, are not) defined as needed.
		In what ways do you find this manual useful?	○ Learning to use the equipment ○ To instruct a class. ○ As a reference ○ Other: ○ As an introduction to the product
☐	☐	Do the illustrations help you?	○ Visuals (are,are not) well designed. ○ Labels and captions (are,are not) clear. ○ Other:
☐	☐	Does the manual tell you all you need to know? What additional information would you like?	
☐	☐	Is the information accurate? (If not please specify with page number and paragraph.)	

Name: _____ Title: _____

Company: _____ Division: _____

Address: _____ City: _____

State: _____ Zip: _____ Telephone: _____ Date: _____

DG-06895

CUT ALONG DOTTED LINE

DataGeneral
Data General Corporation, Westboro, Massachusetts 01580

FOLD FOLD

TAPE TAPE

FOLD FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 26 SOUTHBORO, MA. 01772

Postage will be paid by addressee

Data General

ATTN: Technical Products Publications (C-138)
4400 Computer Drive
Westboro, MA 01581

CUT ALONG DOTTED LINE

DataGeneral users group

Installation Membership Form

Name _____ Position _____ Date _____

Company, Organization or School _____

Address _____ City _____ State _____ Zip _____

Telephone: Area Code _____ No. _____ Ext. _____

1. Account Category

- ☐ OEM
- ☐ End User
- ☐ System House
- ☐ Government
- ☐ Educational

2. Hardware

	Qty. Installed	Qty. On Order
M/600		
COMMERCIAL ECLIPSE		
SCIENTIFIC ECLIPSE		
AP/130		
CS Series		
Mapped NOVA		
Unmapped NOVA		
microNOVA		
Other (Specify) _____		
_____		
_____		

3. Software

- ☐ AOS
- ☐ DOS
- ☐ MP/OS
- ☐ RDOS
- ☐ Other

Specify _____

4. Languages

- ☐ Algol
- ☐ DG/L
- ☐ Cobol
- ☐ PASCAL
- ☐ Business BASIC
- ☐ BASIC
- ☐ Assembler
- ☐ Fortran
- ☐ RPG II
- ☐ PL/1
- ☐ Other

Specify _____

5. Mode of Operation

- ☐ Batch (Central)
- ☐ Batch (Via RJE)
- ☐ On-Line Interactive

6. Communications

- ☐ HASP
- ☐ RJE80
- ☐ RCX 70
- ☐ CAM
- ☐ XODIAC
- ☐ Other

Specify _____

7. Application Description

○ _____

8. Purchase

From whom was your machine(s) purchased?

- ☐ **Data General Corp.**
- ☐ Other

Specify _____

9. Users Group

Are you interested in joining a special interest or regional Data General Users Group?

○ _____

DataGeneral

Data General Corporation, Westboro, Massachusetts 01580. (617) 366-8911

FOLD *FOLD*

TAPE *TAPE*

FOLD *FOLD*

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 26 SOUTHBORO, MA. 01772

Postage will be paid by addressee:

Data General

ATTN: Users Group Coordinator (C-228)
4400 Computer Drive
Westboro, MA 01581

CUT ALONG DOTTED LINE

Data General

users group

Installation Membership Form

Name _____ Position _____ Date _____

Company, Organization or School _____

Address _____ City _____ State _____ Zip _____

Telephone: Area Code _____ No. _____ Ext. _____

1. Account Category

- ☐ OEM
- ☐ End User
- ☐ System House
- ☐ Government
- ☐ Educational

2. Hardware

	Qty. Installed	Qty. On Order
M/600		
COMMERCIAL ECLIPSE		
SCIENTIFIC ECLIPSE		
AP/130		
CS Series		
Mapped NOVA		
Unmapped NOVA		
microNOVA		
Other (Specify) _____		
_____		
_____		

3. Software

- ☐ AOS
- ☐ DOS
- ☐ MP/OS
- ☐ RDOS
- ☐ Other

Specify _____

4. Languages

- ☐ Algol
- ☐ DG/L
- ☐ Cobol
- ☐ PASCAL
- ☐ Business BASIC
- ☐ BASIC
- ☐ Assembler
- ☐ Fortran
- ☐ RPG II
- ☐ PL/1
- ☐ Other

Specify _____

5. Mode of Operation

- ☐ Batch (Central)
- ☐ Batch (Via RJE)
- ☐ On-Line Interactive

6. Communications

- ☐ HASP
- ☐ RJE80
- ☐ RCX 70
- ☐ CAM
- ☐ XODIAC
- ☐ Other

Specify _____

7. Application Description

○ _____

8. Purchase

From whom was your machine(s) purchased?

- ☐ **Data General Corp.**
- ☐ Other

Specify _____

9. Users Group

Are you interested in joining a special interest or regional Data General Users Group?

○ _____

Data General

Data General Corporation, Westboro, Massachusetts 01580. (617) 366-8911

FOLD FOLD

TAPE TAPE

FOLD FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 26 SOUTHBORO, MA. 01772

Postage will be paid by addressee:

Data General

ATTN: Users Group Coordinator (C-228)
4400 Computer Drive
Westboro, MA 01581

Data General

Data General Corporation, Westboro, Massachusetts 01580

069-400200-00