System Commands
Reference

Data Genera

Notice

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.

Ordering No. 093-400051

Revision History:
Original Release - June 1982
Rev. 01, December 1982

Effective with:
MP/AOS Rev. 1.00

Contents

Preface

1. System Overview

2. Programming with MP/AOS

3. File Management

4. Process Management

5. Memory Management

6. Process Debugging and Runtime Performance Monitoring

Figures

Tables

Preface

This manual is intended to serve experienced system programmers who want to know all the details of MP/AOS system software.

The manual contains twelve chapters. Chapter 2 gives a general view of programming with MP/AOS. Chapters 3, 4, and 5 deal with the management of files, process, and memory, respectively. Debugging and histogramming are covered in Chapter 6. Chapters 7 and 8 discuss interprocess communication and multitasking. Input and output and user device support are discussed in Chapters 9 and 10. The last two chapters contain a list of miscellaneous calls and a dictionary of system calls and library routines.

There are eleven appendices and an index for reference.

Related Manuals

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

MP/AOS Concepts and Facilities (DGC No. 069-400200) provides a concise but thorough introduction to the MP/AOS operating system for users who want to assess the system's advantages.

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 ECLIPSE-line computers.

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 folowing 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, minimizing effort and errors in program development;
- FIND, which identifies occurrences of patterns 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 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 extensions to implement 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 with special extensions.

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).

Conventions and Abbreviations

In format descriptions and examples we use the following conventions:

Format	Descriptions
COMMAND SYSTEM CALL	This typeface is used to indicate mnemonics for commands, system calls, and instructions; e.g. ?ALIST.
argument	Lowercase *italic* is used to represent a command's or an instruction's argument when that argument is a generic term. In your program, you must replace this symbol with the exact code for the argument you need; e.g. file1.
[*optional*]	This typeface, lower case *italic* and brackets denote an optional argument. Optional command switches may appear within brackets as well. If you use the argument or switch, do not write the brackets into the code.
arg1 | *arg2*	This typeface, lower case *italic* and a vertical bar (|) denote that you have a choice between arg1 or arg2.
↵	Represents a New-line character.
CR	Represents a Carriage Return character.

Examples

All programming examples appear in the following typefaces:

the material to be typed by the user appears as:

```
?PROC
```

the program's response appears as:

```
ERBTL
```

System Overview

The MP/AOS system is a general purpose operating system for 16-bit DGC ECLIPSE processors available in both 15" and 7"x 9" packaging. Processors must have the Floating Point Instruction option and the Character Instruction set. Currently, MP/AOS runs on the following ECLIPSE processors: S/120, S/130, S/140, C/150, C/330, S/250, C/350, S/20.

MP/AOS retains substantial compatibility with MP/OS, a single-user system for microNOVA, ENTERPRISE, MPT, MBC, and NOVA4 computers. 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 ECLIPSE line computers.

MP/AOS provides sophisticated facilities such as multiprogramming, multitasking, interprocess communication, flexible user management of system resources, the ability to access more than 32K words 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.

Programs communicate with the operating system through *system calls* that you place in the program code. This manual describes the operating system's facilities and the system calls that apply to them.

Programming with MP/AOS

Before beginning to work with the MP/AOS operating system, you must first *bootstrap* the system, i.e., bring it into memory. Since the bootstrap procedure varies depending on the processor used, it is necessary to refer to the appropriate Principles of Operation manual for your CPU. A detailed discussion of bootstrapping also appears in *Loading MP/AOS* (DGC No. 069-400207).

Shutting Down the System

Any one of the following procedures results in an orderly system shutdown ensuring no loss of data:

a BYE command issued from the initial CLI
a ?BOOT system call issued from the initial process
termination of the initial process

System Calls

MP/AOS supports a wide variety of *system calls*, command macros which call on predefined system routines. In assembly language, system calls are coded in the user program just as instructions are.

MP/AOS system calls allow the user to

- create and manage processes
- manage the logical address space
- manage dynamic memory segments
- establish and perform interprocess communication
- create and maintain disk files and directories
- perform file input and output including direct segment I/O
- create and manage a multitasking environment
- define and access user devices
- perform data channel or BMC (Burst Multiplexor Channel) input and output with user devices
- define and manage custom debugger routines
- define and manage custom histogrammer routines

In assembly language, code system calls in the source program as macros that begin with a question mark. Each macro is expanded at assembly time. Each system call macro name is associated with a number. A complete list of system call mnemonics and their numbers is included in parameter file SYSID.SR, which is distributed with the release package. See Appendix E for more on parameter files.

A special system call (?EQT) offers users the option of setting up system calls at runtime by specifying the desired system call number and option in AC3 and setting up AC0 through AC2 as defined for the particular call to be executed.

This manual discusses the system calls in functional categories with a chapter for each category. A detailed description of each system call appears in the alphabetized "Dictionary of System Calls and Library Routines" in Chapter 12.

When generating an MP/AOS system you specify the maximum number of concurrent system calls to be supported by that system. The interactive SYSGEN utility is described in *MP/AOS System Generation and Related Utilities* (DGC No. 069-200206).

Error Codes

Except where noted, you must reserve two return locations for each system call: an *exception error return*, and a *normal return*. After the system has executed the call, MP/AOS passes control either to the error return or to the normal return, depending on the call's outcome.

In either case, on return, accumulator 3 (AC3) contains the current contents of the frame pointer. On an exception return, AC0 contains an unsigned 16-bit value representing the *exception condition code* (error code) indicating the reason for the call's failure. All other accumulators contain the values they had on input, unless otherwise noted.

A unique text string is also associated with each error code. The CLI (Command Line Interpreter) returns this string when the error occurs during the execution of a CLI command. Use the ?ERMSG library routine to read the text string associated with the error code during the execution of the program.

The "Errors" list in the individual description of each system call (Chapter 12) gives the most likely exception condition code mnemonics and messages for that particular call. A complete list of fatal and booting error codes is contained in your release package.

The system provides a file, ERMES, containing all the currently defined error codes and their corresponding mnemonics and text messages. There are 200_8 groups of exception condition codes for the operating system, the utilities, and the other programs running in the system, including user programs. Data General Corporation reserves code groups 0 through 77_8 for the system. You can define the remaining groups, numbered 100_8 through 177_8.

To create a new ERMES error message file with a structure like that of the supplied ERMES, but with different contents, create a source file allocating an unused code group and insert your own series of codes and messages. After assembly, bind with a /ERMES switch.

If you wish your new ERMES file to include any of the DGC-supplied error codes, set your searchlist to allow access to ERMES_OBS, the error message object files supplied on your release media. You can then bind the desired ERMES_OBS files along with your own to generate a combined ERMES file.

System Call Options

Some system calls have *options* you may specify to modify the calls' actions. Options are specified by two- or three-letter abbreviations, which you code after the call's mnemonic in the program. For instance, if you want to create a disk file with the ?CREATE call and you wish to delete an existing file with the same name, use the delete (DE) option by coding ?CREATE DE. You can specify more than one option by separating the options' abbreviations with commas, for example, ?OPEN CR, AP, which creates a file if none exists and opens it for appending.

Nonpended Calls

Some system calls, notably those that perform I/O, can take a relatively long time to execute. Normally your program (or the calling task in a multitasked program) is suspended from running during this interval, resulting in a loss of potentially useful processor time. *Nonpended* system calls eliminate this waste.

Specify a nonpended call by coding the NP option on any system call allowing it. When you execute the call, instead of suspending your task, the system creates a new task and assigns it the job of executing your call. The task which issued the nonpended call is free to continue operation. AC2 will contain the task identifier of the system task executing your I/O. To avoid error when using nonpended calls, be sure to specify an appropriate number of additional tasks (one for each concurrent nonpended call) when you create the process. See the ?PROC system call description in Chapter 12.

You cannot immediately assume that the results of the system call are valid; for instance, if you read data with a ?READ NP system call, you must still wait for the data to arrive before you can operate on it. However, you can perform other types of computation while waiting for the new data.

To determine when the nonpended call is complete, you must execute the ?AWAIT system call. It enables you either to check the call's progress or to suspend your program until the call is complete. You must issue an ?AWAIT call to obtain the results of every nonpended system call you execute; otherwise system memory space (a task control block) is wasted.

Accumulator Usage

System calls generally require arguments called *inputs*, which your program must place in the proper accumulators before executing the call. Some system calls also return *outputs* in accumulators. Only AC0, AC1 and AC2 are used for inputs and outputs; the system always sets AC3 to the value of the frame pointer upon return from a call. Any accumulators not used for outputs are returned to your program unchanged.

MP/AOS system calls and library routines observe the following conventions for accumulator usage.

- Input/output calls use AC0 for the I/O channel number.
- Multiprogramming and multitasking calls use AC2 for the process or task identifier.
- Calls that reference files use AC0 for the byte pointer to the pathname.

- Calls that require packets use AC2 for the packet address.
- Error codes are returned in AC0.

Byte Pointers

Before issuing many of the system calls, you must load one or more of the accumulators with input values, such as *byte pointers.*

An ECLIPSE computer *word* is 16 bits in length; its bit positions are numbered left to right, from 0 to 15 inclusive. A *byte* is 8 bits in length. A byte string consists of a sequence of bytes, packed left to right in a series of one or more words.

The system call descriptions use unique mnemonics for the high-order and low-order portions of 16-bit values. The term *high-order* refers to the 8 most significant bits, i.e., bits 0 through 7. The term *low-order* refers to the 8 least significant bits, i.e., bits 8 through 15.

A *byte pointer* consists of a single word with two fields. The left field consists of bit positions 0 through 14, and it contains the address of the word containing the selected byte. The right field consists of the bit position 15. When the state of this bit equals 1, the pointer selects the low-order (least significant) byte of a word, i.e., bits 8 through 15; when the state of this bit equals 0, the pointer selects the left (high-order, or most significant) byte of a word, i.e., bits 0 through 7.

To point to a byte in a word whose address you have defined as a variable (V), the value V*2 serves as a byte pointer to the left byte, and V*2+1 points to the right byte.

Packets

Some system calls require or return more information than the accumulators alone will hold. In this case, additional arguments are passed in a *packet.* A packet is a block of consecutive words in an address space. The system uses these words to obtain input specifications and/or to return output values.

The number of words in a packet depends on the particular call; there is a mnemonic for each packet size. The first word of every packet contains a number indicating the packet type; the system checks this number for validity when handling the call.

There is also a mnemonic for each packet type. Using the mnemonic instead of the current value assigned to it ensures that even if the value is redefined, the call executes correctly, provided you reassemble your program. The mnemonics and their current values appear in parameter file OPARU.SR. (See Appendix E.)

The system uses the data you supply in a parameter packet to decide how to execute a call. The location of data in a packet determines its interpretation. There are two ways of setting up a parameter packet: with *absolute addressing* or *offset addressing*.

Offset addressing (or offset words) consists of system-defined parameters, also listed in OPARU.SR, which reference the words of the parameter packet. The packets described in the system call dictionary, Chapter 12, all use offset addressing. This ensures that if the packet is redefined in a future release of MP/AOS, the program will still run correctly if reassembled, because Data General will insure that the word offsets still correspond to the appropriate data, regardless of their location in the packet.

Some parameter packets contain *flag words*, in which each bit has a special meaning. These bits are set with *bit masks,* system-defined parameters listed in OPARU.SR, each equal to a single set bit.

NOTE: *You must set all reserved words in a parameter packet to zero.*

See ?FSTAT in Chapter 12 for an illustration of a packet.

Stacks

Each MP/AOS task that issues library calls must have a *stack* area in memory for library calls to use. The stack size must be equal to or greater than the value of the mnemonic ?STKMIN. You can initialize the stack control words by using the assembler's .LOC directive. When a program starts, the contents of locations 40_8 and 41_8 are the stack pointer and frame pointer, respectively. You must also initialize location 42_8 with the stack limit, and you may initialize location 43_8 with the address of a *stack fault handling routine.*

The system calls the stack fault handling routine if your program attempts to exceed the specified stack limit. The routine may perform functions such as allocating more memory or simply shutting down the program. Before calling the routine, the hardware pushes five words onto the stack, whose contents (in the order pushed) are

- the accumulators AC0 through AC3
- a word containing the carry in bit 0 and the contents of the program counter (where the overflow occurred) in bits 1-15.

NOTE: *Since the system handles stack overflow by pushing more words onto the stack, make sure that the stack is actually five words larger than the size you specify in the stack limit word. Otherwise, part of your program code may be destroyed during the handling of the overflow. You should also allow for any stack space the overflow handling routine itself may need.*

If your program uses multitasking, each task must have its own stack area. The stack pointer, frame pointer, stack limit and stack fault handler address are specified in the ?CTASK packet. In this case , the system maintains the stack control words so that each task always has its own unique values.

Naming Conventions

All symbols containing a **?** are reserved for the system's use. All symbols starting with ER are reserved for error codes.

Mnemonics

All symbols, such as error codes and offsets in parameter tables, are referred to by their defined mnemonics instead of their numeric values.

Mnemonics which represent *status flags* have values that set the named bit to 1 and all other bits to 0. Thus you can use the mnemonic's value in a logical AND to determine the *flag setting*. To set several flags at once, code an assembler expression containing the sum of several mnemonics.

All mnemonics for system calls, library routines, error codes and other symbols used in this manual are defined in the file MASM.PS, the assembler's permanent symbol table. Many of them are defined in the user parameter file, OPARU.SR, a listing of which appears in Appendix E of this manual. Refer to the parameter file to determine the value of a symbol; usually, though, you can use the mnemonics in your program without knowing their values.

It should be stressed that parameter file values are revision-dependent; the user is urged to check release notices for the latest information update.

Library Routines

MP/AOS provides a number of convenient functions implemented as library routines rather than system calls. A list of currently available routines appears in Appendix J. Chapter 12, "Dictionary of System Calls and Library Routines" identifies and describes each routine individually.

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

- suspending the operation of a program for a specified time period
- providing several timing facilities to support real-time operations
- setting the searchlist
- reading a message from an MP/AOS error message file.

Program Revision Number

The system maintains a revision number in every program file to help you track different versions of a program. This number consists of a major and a minor revision number; each may range from 0 to 255. Set the number with the .REV assembler directive or the Binder /REV=*value* keyword switch; read the number with the ?INFO call; and use the CLI REVISION command to read or set the number.

Overlays

Under MP/AOS, overlay loading and release is accomplished with library calls. The MP/AOS overlay facility is flexible: the exact distribution of overlay blocks is not specified until bind time; hence, no program modification is needed to experiment with different strategies. This makes it easy to reorganize overlays for greatest efficiency.

Overlays are discussed further in Appendix F.

File Management

The MP/AOS file system provides the user with simple, efficient ways to communicate with input/output devices and to store and retrieve data in files. Because all devices and files are handled by the same system calls, it is easy to write device-independent programs.

Basic Organization

An MP/AOS file is either an I/O device, such as a printer, or a collection of data stored in a disk file. Since both kinds are handled identically, we use the term *file* to refer to either.

A file is referenced by its *filename*, a string of one to fifteen characters. The mnemonic ?MXFL contains the value for maximum filename length. Legal characters in filenames are

- the letters a to z and A to Z; (You can use upper- and lower-case interchangeably; the system considers them equivalent and uses only upper-case internally.)
- the digits 0 to 9;
- the punctuation marks ?, $, _ (underscore) and .(period).

Hierarchical File Structure

In general, a file's contents are entirely user-defined; however, several types of files have special functions. In particular, there is a type of file called a *directory*, which contains other files. Three special directories used by the system (the device, root, and system working directory) are discussed below.

Any file in a directory may itself be a directory containing other files. A directory within another directory is called a *subdirectory. Nesting* of directories may continue indefinitely in this manner.

Files within directories are referenced by using the : character. For example, X:Y references a file named Y in a directory named X. An expression of this form is called a *pathname*. Pathnames are explained in detail later in this chapter.

The Device Directory

The device directory is the *highest* directory in an MP/AOS *system:* it contains all others. This directory has the special symbol @ as its filename.

The filenames in the device directory correspond to all the input/output devices in the system. To reference a device, use its name prefixed by @. The name may be followed by a one- or two-digit unit number. Typical device names are @LPT for a line printer or @DPD0 for a disk drive.

Since the device directory contains all I/O devices including disks, it cannot be contained on any device. Thus, the device directory is unique in the system in that it is not physically represented on any disk. It is actually a table in MP/AOS memory space and cannot be accessed via the ?DIR system call.

Root Directories

Every disk device has a *root* directory which is the highest directory on the *device.* The root directory and its subdirectories contain all other files on a disk. The root directory is referenced by appending a : to the device name, e.g., @DPD0:.

The System Working Directory

The system working directory (:?SYSDIR) is used by the system to contain various special purpose temporary files and user break files. (Break files are explained in detail in Chapter 4.)

Caution should be used in modifying or deleting any files other than break files in the system working directory, as this may result in unpredictable system behavior.

System Master Device

One disk unit in every MP/AOS system is the *system master device*; i.e., the device from which the operating system was bootstrapped. The MP/AOS system program files and many other commonly-used files reside in this unit. For ease of reference, MP/AOS accepts the : character as a prefix which refers to the root directory of the system master device. For example, if your system's master device is @DPD0, then the pathname :CLI.PR is equivalent to @DPD0:CLI.PR.

Pathnames

The system allows one filename to be used simultaneously for several files in *different* directories, but filenames must be unique within any one directory. The capacity to reference any file uniquely is provided by *pathnames.* As its name suggests, a pathname represents a path through the directory structure to a particular file.

A pathname consists of a series of filenames separated by colons (:). Pathnames may be up to 127 characters long. The ?MXPL parameter specifies maximum pathname length. All the files named except the last must be directories; each directory named must be a subdirectory of the preceding one. For example, the pathname A:B:C references a file called C in subdirectory B of directory A.

A pathname beginning at the device's root directory is called a *fully-qualified* pathname, since it is guaranteed to identify only one file in the entire system. An example of a fully-qualified pathname is @DPD0:A:B:C.

When you supply a pathname as an argument to a system call or library routine, it must be terminated by a null (zero) byte. The system always uses this format when passing pathnames to your program. Remember to allow sufficient buffer space to hold the filename and the terminator byte whenever you use a call that returns a file- or pathname.

Figure 3.1 shows a typical fragment of an MP/AOS file system. The device directory contains several I/O devices including one disk drive. The fully-qualified pathnames of these devices are shown in bold type.

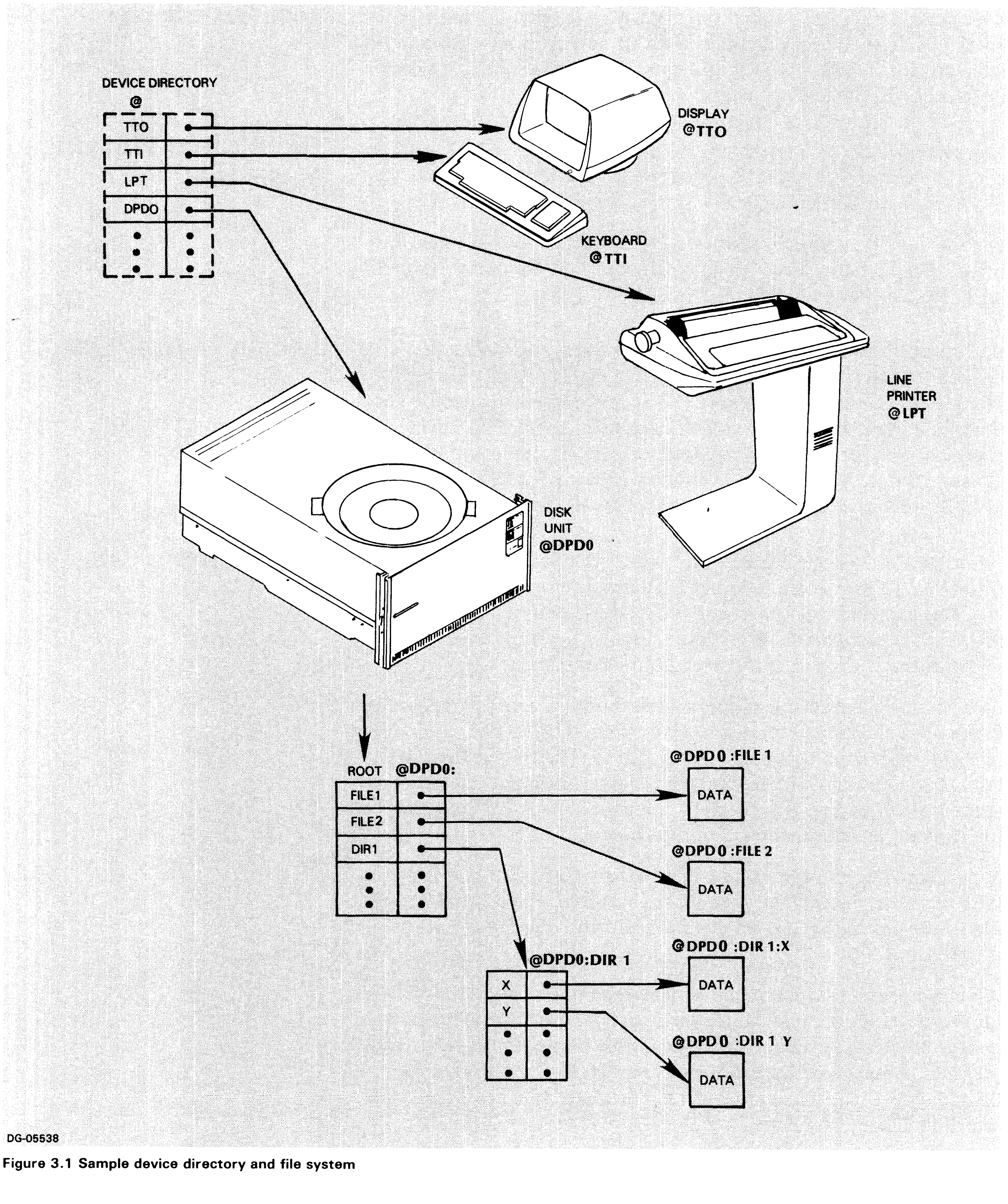


Figure 3.1 Sample device directory and file system

The disk's root directory contains three files named FILE1, FILE2 and DIR1. DIR1, a subdirectory of the root, contains two files called X and Y. Their fully-qualified pathnames are also shown in bold type.

The Working Directory

An MP/AOS system typically contains many more files than Figure 3.1 shows. As directory structures become more complex, pathnames become longer and more cumbersome. To reduce the necessity of using long pathnames, the system assigns a *working directory* to every program. The working directory may be thought of as your current location in the file structure.

Whenever you reference a filename or pathname that is not fully qualified, the system looks for the file in your working directory. This enables you to use simple filenames instead of pathnames and confines all file activity to the working directory. A pathname such as A:B refers to a file called B in subdirectory A of your working directory.

Since users typically create a directory for each project, this concept allows related files to be kept together. You can change your current working directory at any time with the ?DIR system call and determine your current working directory with the ?GNAME call. You can also perform these functions with the CLI DIR command.

The Searchlist

Sometimes it is inconvenient to confine all one's work to a single directory. For this reason, the system provides a *searchlist*, a concise method of referencing multiple directories. The searchlist is simply a list of pathnames of directories. If you use a filename that is not fully qualified and if the named file is not in your working directory, the system searches all the directories in your searchlist before determining that the file does not exist. The system searches all paths in this manner, except for those specified in a ?CREATE, ?DELETE or ?RENAME call. If the same filename exists in more than one of the directories in the searchlist, the system uses the file appearing in the first directory it encounters.

You can read your searchlist with the ?GLIST system call, and you can clear or extend your searchlist with the ?ALIST call. There is also a convenient ?SLIST library routine that establishes your searchlist with one call and a CLI SEARCHLIST command that reads or creates the searchlist. When the system is started up, it sets your searchlist to contain only the system master device's root directory, :.

Pathname Prefixes

The use of the @ and : characters in pathnames has already been explained. Two other characters may be used as *prefixes*; i.e., they may appear only at the beginning of a pathname.

The = character is equivalent to the pathname of the *current working directory.* Use this character to reference files in the working directory explicitly; the searchlist is not used if the file is not found. The = character alone can also be used as the name of the current working directory.

The ↑ (uparrow) character, typed as ^ and echoed as either ^ or ↑, refers to the *parent* directory, i.e. the one containing the current working directory. For instance, if your current working directory is @DPD0:A:B and you want to reference the file @DPD0:A:XYZ, you can use the pathname ↑XYZ. You can also use several ↑s in sequence: for instance, to reference @DPD0:X, you could use ↑↑X.

NOTE: *A pathname beginning with = or ↑ is not, strictly speaking, a fully-qualified pathname, since the exact meaning of the pathname depends on the current working directory. However, such a pathname is like a fully-qualified pathname because it specifies a directory; hence the searchlist is not scanned.*

Links

Links simplify file referencing by eliminating the need to type lengthy pathnames. A *link* is a file of type ?DLNK containing a pathname or a partial pathname. Generally, when a linkname appears in a pathname, the system resolves it by replacing the linkname with the contents of the link file. The exceptions are discussed below. Links may contain up to 62 bytes.

Normally a link is resolved when it appears in a pathname. If, however, a link is the last or the only filename in a pathname used as argument to ?CREATE, ?DELETE, or ?RENAME system calls, the link will not be resolved. This permits link creation, deletion, and renaming.

If, for example, the pathname
A:B:file_C

is used as an argument to ?DELETE and if B is a link equivalent to D:E, then the pathname is resolved to

A:D:E:file_C

and file_C is deleted.

If, however, the argument to ?DELETE is pathname
A:B

then link B itself is deleted, not directories D and E.

Creating, renaming, and deleting links can also be done by means of CLI commands.

NOTE: *The system does not validate the contents of a link entry until that entry is used in a pathname resolution. Thus it is possible to create a link entry pointing to a nonexistent pathname or containing illegal filename characters. The system returns an error, however, if there is an attempt to use such a link in a pathname.*

You can obtain information on the link entry (rather than its resolution) with the ?FSTAT system call.

If the link contents begin with a prefix (@, :, =, or ^), pathname resolution begins at the directory indicated by the prefix. If the link's contents do not start with a prefix, pathname resolution continues at the point in the directory hierarchy where the link entry was encountered.

File Element Size

The system allows you to optimize disk file organization by controlling the size of *file elements.* A file element consists of one or more 512-byte disk blocks physically contiguous on the disk surface. The system allocates and deallocates file space in elements rather than blocks.

You specify a file's element size when creating the file. A large element size means that data in a file is organized in a number of large groupings. Reading or writing the file can be done more efficiently, since the disk heads do not need to be continuously moved around the disk to find the proper data. Small element sizes give the system greater freedom in allocating disk blocks and result in relatively less unused (wasted) space. Choose the element size that offers the best compromise between speed and efficient use of space.

File Types and Attributes

One specific type of file, the directory, has already been mentioned. Every file in the system has a 16-bit number that defines its type. You can specify file type when creating a file with the ?CREATE call; read file type with the ?FSTAT call.

Assign any meanings to these file types that you find useful. Table 3.1 summarizes the available file types.

Mnemonic	Meaning	
?DDIR	Directory	
?DSMN to ?DSMX	Range of values for files used by the system:	
	?DBPG	bootable (stand-alone) program file*
	?DBRK	program break file
	?DIDF	MP/ISAM data file
	?DIXF	MP/ISAM index file
	?DLIB	library file
	?DLNK	link file
	?DLOG	System log file
	?DMBS	MP/BASIC save file
	?DOBF	object file
	?DOLF	overlay file
	?DPRG	program file
	?DPST	permanent symbol table (used by assembler)
	?DSTF	symbol table file
	?DTXT	text file
	?DUDF	general-purpose data file
?DUMN to ?DUMX	Range of values reserved for users	

Table 3.1 File types

**Currently not bootable under MP/AOS.*

The system also maintains an *attribute word* for each file. The right half (bits 8-15) of this word is used or reserved by the system. The left half (bits 0-7) is reserved for the user. As with file types, you may assign any meanings you wish to these bits.

You can read a file's attributes with the ?GTATR call and change them with the ?STATR call. Table 3.2 summarizes file attributes.

Mnemonic	Meaning
?ATPM	Permanent: the file may not be deleted or renamed while this bit is set to 1. Set by the system for directories and root directories of disks.
?ATRD	Read protect: this file may not be read.
?ATWR	Write protect: this file may not be written. Set by the system for directories and root directories of disks.
?ATAT	Attribute protect: the attributes of this file may not be changed. Set by the system for devices and root directories of disks only.

Table 3.2 File attributes

When the system creates an entry for a new file in a directory, the current time and date are associated with the new filename. Subsequently, the system updates the time and date to reflect the last occasion on which the user accessed or modified the file. This information can be obtained via the ?FSTAT system call: packet double word ?FTLA in ?FSTAT returns the date and time the file was last accessed; packet double word ?FTLM returns the date and time the file was last modified.

System Call/Library Routine Summary

Table 3.3 summarizes MP/AOS system calls and library routines for file management.

Mnemonic	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 file status including type, attributes, size; can be used to retrieve link contents.	CH (file is open on specified channel number) LNK (do not resolve links)
?GLIST	Get current searchlist	
?GNAME	Get fully-qualified pathname; scans searchlist if necessary	CH (file is open on specified channel number) PID (get pathname of process) PR (get pathname of calling program)
?GNFN	Library routine: get next filename; retrieves names of files contained in a directory	
?GTATR	Get file attributes and file type	CH (file is open on specified channel number) LN (return file byte length)
?RENAME	Rename a file; can be used to move a file to a new directory	DE (delete existing filename)
?SLIST	Library routine: set the searchlist	
?STATR	Set file attributes	CH (file is open on specified channel number)

Table 3.3 Summary of file calls and library routines

Input/Output

Data transfers between your program and a device on file are detailed in Chapter 9, "Input and Output".

Process Management

MP/AOS multiprogramming and multitasking capabilities enable development of a wide range of applications systems. Process servicing is user-defined with a priority system. Extensive system calls provide for complete user control of process space, scheduling, and process I/O. This chapter describes the facilities available for managing processes under MP/AOS. The MP/AOS Interprocessor Communication (IPC) facility which provides process synchronization and interaction is described in Chapter 7.

Process Concepts

A *process* is an environment within which a program can execute in parallel with other programs. The process defines and controls the real time resource management performed by MP/AOS for its users. These resources include the following:

- the program to be executed for this process
- a message (2048) bytes which can be passed to the process
- pathnames of devices or files to be opened on channels 0 and 1
- an initial searchlist setting
- an initial directory setting
- initial process priority
- maximum number of channels the process can use
- maximum number of tasks that can execute within the process
- maximum amount of memory the process can use
- maximum number of memory segments the process can attach to
- maximum number of overlay nodes

The process thus represents facilities allocated to the executing program, as well as constraints within which that program is confined. The constraints indicate how many system resources are required for the execution of a given process, and whether or not the system is capable of executing it.

In support of its multi-process environment, MP/AOS permits a dynamically varying number of independent processes. You specify system configuration parameters during the interactive SYSGEN dialogue. You can, for example, initiate a maximum of 16 individual processes that can contain a total of 256 tasks executing up to 255 concurrent system calls.

MP/AOS processes are *concurrent* and *resident:* all processes run in parallel, and each process resides in main memory. The system allocates CPU control to each process according to process *priority* with optional support for time-slicing for equal priority processes. Priority is user-assigned.

A process can define its own environment, communicate with other processes by means of interprocess communication calls (IPC), and control process scheduling. MP/AOS provides a separate process debugger that can initiate and block a user program for the detection of potential runtime errors during program development.

The Initial Process

The *initial process* is created by the system when MP/AOS is initialized. The specific program executed by the initial process is a system generation parameter; you may specify any program file for this purpose. A typical program running as the initial process might be the Command Line Interpreter (CLI.PR), the interface between the user at the console and the system.

The initial process is the only process capable of issuing a ?BOOT command to shut down the system. Program termination from swap level 1 of the initial process also causes the system to shut down.

Concurrent Processes

MP/AOS does not have a process hierarchy.

Once a concurrent process is created, it has no further relationship to, or dependency on, the process creating it.

Resident Processes

All processes are *resident;* that is, they reside permanently in main memory until terminated. Resident processes cannot be displaced by any other process even if they are performing lengthy I/O or even if they are blocked for an appreciable amount of time.

While dynamic swapping of programs is not supported, programs may be initiated with the ?EXEC system call to execute within the same process. When the new program is ?EXECuted, the initiating program is *swapped out* (written to a disk file). The new program is said to run at the next higher *swap level,* while retaining the same process ID as the initiating program.

If ?EXECuted with the chain option, the new program runs at the *same* swap level as the initiating program. The calling program state is not saved. ?EXEC is further discussed in a later section.

Process Priority

The *priority* of a newly initiated process must be specified by the creating process. The value of the priority argument can range from 0 to 255, with 0 the highest priority and 255 the lowest.

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 receives system services regardless of the priority of its current task. A process' task is scheduled if it is ready to run (i.e., 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 opt for *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 run 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 (i.e., when all its tasks pend) and not before. The time-slicing feature does not affect task scheduling.

A program can retrieve the priority of another process with the ?EINFO system call. To retrieve its own process priority, a program can issue either ?EINFO with Process ID = 0 in AC0, or ?MYID with the PRC option. To modify its own or any other process priority, a program issues the ?PRI call with the PRC option.

Process Identifier

Whenever a process is initiated (with a ?PROC call), the system assigns it a *process identifier* (PID, or process ID). The process ID, a unique value in the range of 1-65535, is assigned in ascending order. When PID 65,535 has been assigned, the next ID to be assigned is 1, or the lowest number not in use by a currently executing process.

A process can retrieve its own PID (?MYID with the ?PRC option). To determine the PIDs of other processes currently in the system, use the ?GTPID call, which requires a buffer for the PID information. Note that ?GTPID provides a snapshot of the system at the moment of calling; a specific process might not exist by the time the call returns.

Programs activated with an ?EXEC call retain the PID of the creating process. (See Figure 4.2.)

Creating a Process

The ?PROC call is used to create a process, and any process can create other parallel processes at will by issuing a ?PROC call for each one. Process creation can be performed from any swap level.

?PROC takes a packet of specifications defining the resources allocated to the new process.

New processes immediately begin executing, subject to process scheduling. Each created process exists independently until it terminates itself or is terminated by another process.

Changing Programs Within a Process: ?EXEC

Within a process MP/AOS offers a push/pop mechanism so that a process can change its environment without losing its previous state within a particular program. This allows the user to operate at different levels with different programs, while remaining within the same process. The number of processes required for an application can thus be reduced.

The ?EXEC call changes the program that a given process is running, while retaining the process state (i.e., the contents of the task control blocks (TCB's) for each task, the channels, impure memory, current segment mapping and relationships to it, the environment, and information needed for restoring overlays).

NOTE: *The CL option to the EXEC call can be used to close all channels for the executed process except the standard input/output channels, ?INCH and ?OUCH.*

When a process is created, it runs a program at swap level 1. When that program issues a ?EXEC, it is swapped out, and the program replacing it runs at level 2. A program created by that program runs at level 3, and so on, up to a maximum swap level of eight. Figure 4.1 illustrates program swapping.

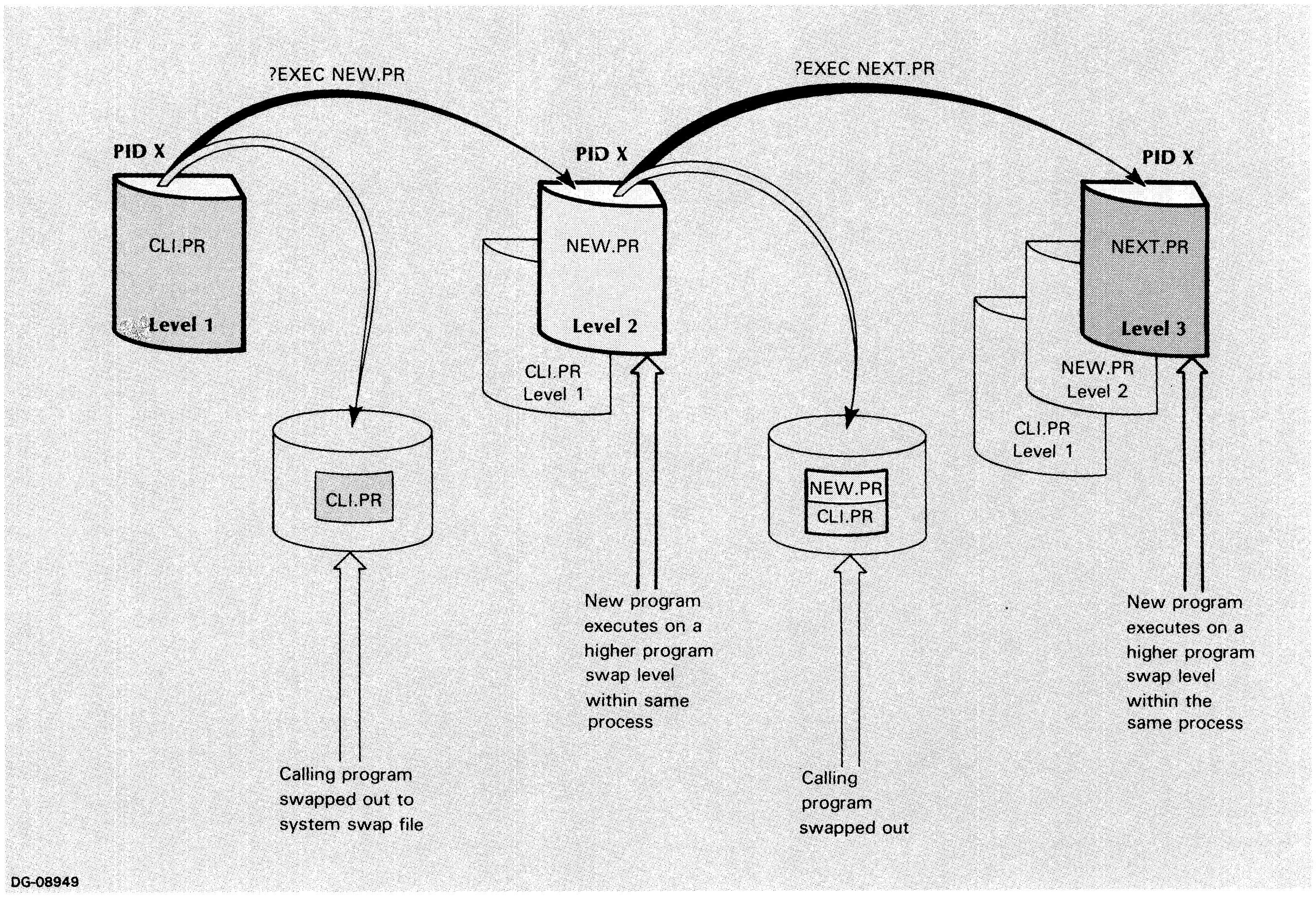

?EXEC NEW.PR
?EXEC NEXT.PR
PID X
CLI.PR
Level 1
PID X
NEW.PR
Level 2
CLI.PR
Level 1
PID X
NEXT.PR
Level 3
NEW.PR
Level 2
CLI.PR
Level 1
CLI.PR
NEW.PR
CLI.PR
New program executes on a higher program swap level within same process
New program executes on a higher program swap level within the same process
Calling program swapped out to system swap file
Calling program swapped out
DG-08949

Figure 4.1 Program swapping via ?EXEC

The new program has the same process identifier number (PID) as its calling program. When a subsequent program executing on a higher swap level terminates, the last swapped-out (not chained) program is reactivated. As used in MP/AOS, the term "parent program" pertains to the push level relationships just described and has no hierarchical connotations. That is to say, a parent program is merely the program on the next numerically lower level to the current program.

To change the program that a process is running while retaining only the process *level* rather than the process state, use the ?EXEC call with the *chain* option. This procedure, called *chaining* to a new program, speeds up the switch by overwriting the calling program with the new one, eliminating the time needed to swap out the creating program. The new program retains the same swap level as the calling program. Figure 4.2 illustrates swap and chain options.

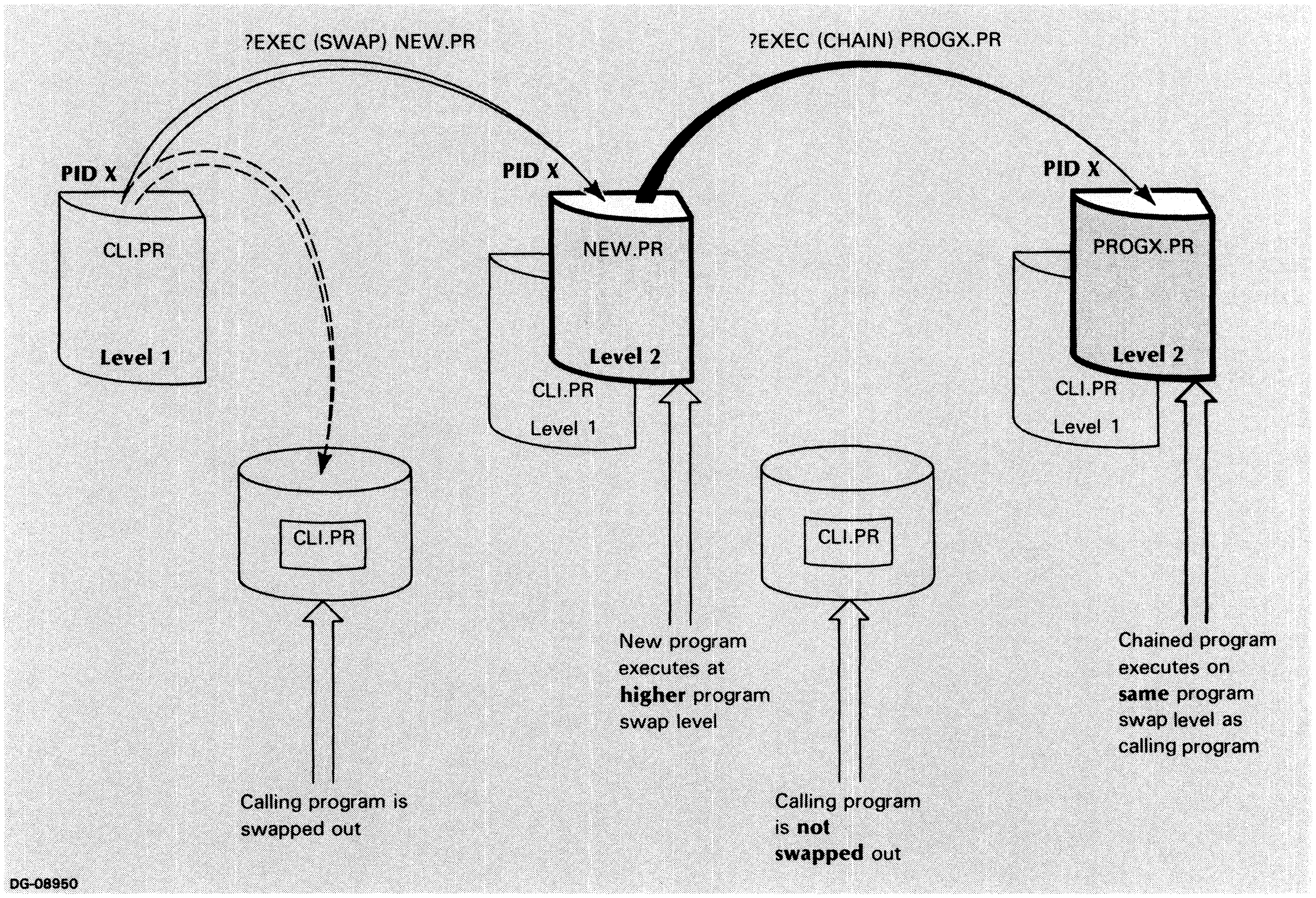

?EXEC (SWAP) NEW.PR
?EXEC (CHAIN) PROGX.PR
PID X
CLI.PR
Level 1
PID X
NEW.PR
Level 2
CLI.PR
Level 1
PID X
PROGX.PR
Level 2
CLI.PR
Level 1
CLI.PR
CLI.PR
New program executes at **higher** program swap level
Chained program executes on **same** program swap level as calling program
Calling program is swapped out
Calling program is **not swapped** out
DG-08950

Figure 4.2 ?EXEC with swap and chain options

When a chained program terminates, it does not return to its calling program; instead, it reactivates the last *non-chained* program, regardless of the number of chained programs between them.

If no swapped-out programs exist, (i.e., the chained program is executing on swap level 1), termination of the chained program causes the process to terminate as well.

Terminating a Process

The ?RETURN or ?KILL system calls allow a program to terminate itself. ?KILL also allows a program to terminate another program running within a different process. The effect of either call is identical, depending on the swap level of the terminated program and on the process within which that program was executing. For example, either call causes the system to shut down if the terminated program is running from swap level 1 within the initial process.

A program may be terminated regardless of whether the process it is running in is blocked or unblocked.

?RETURN enables the terminated program to return a message of up to 2,047 bytes to its creating program. The message is passed in CLI format if that format is specified by the terminating program.

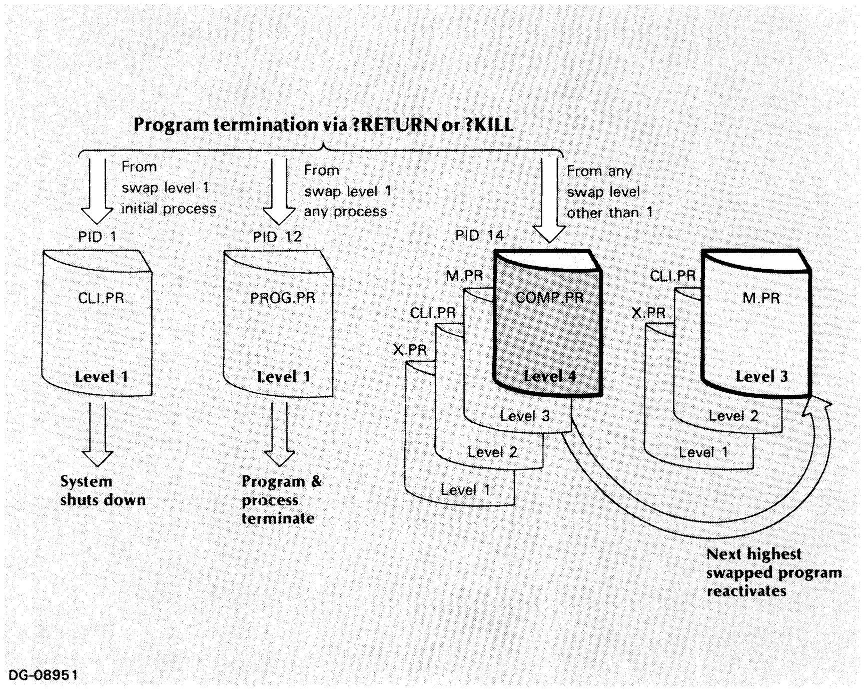
Both ?RETURN and ?KILL allow the creation of a *break file* for later perusal.

Figure 4.3 illustrates the effect of ?RETURN and ?KILL.

If the terminated program was running

- on swap level 1 within the initial process, the system is shut down.
- on swap level 1 but not within the initial process, both the program and the process within which it was running terminate.
- on a swap level higher than 1, the program at swap level n-1 (i.e., the last non-chained program) reactivates.

When an ?EXECuted program is terminated via ?KILL, the parent program receives an ERKIL error code when it is reactivated. (The ERKIL code reads ***Reactivate parent process if current process was terminated by ?KILL and was executing at level other than 1.***)

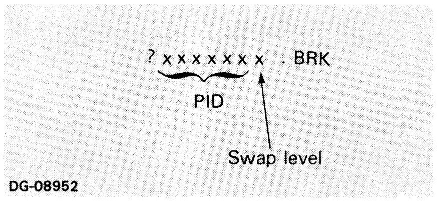

Program termination via ?RETURN or ?KILL
From swap level 1 initial process
From swap level 1 any process
From any swap level other than 1
PID 1
PID 12
PID 14
CLI.PR
Level 1
PROG.PR
Level 1
M.PR
CLI.PR
X.PR
COMP.PR
Level 4
Level 3
Level 2
Level 1
CLI.PR
X.PR
M.PR
Level 3
Level 2
Level 1
System shuts down
Program & process terminate
Next highest swapped program reactivates

DG-08951

Figure 4.3 Effect of ?RETURN and ?KILL

Break Files

When a ?RETURN call or a ?KILL call uses the BK option, MP/AOS writes the information about the terminating program and its state into a *break file* in the system working directory (:?SYSDIR). The name of the break file is composed of a question mark (?) followed by seven digits and a .BRK extension. See Figure 4.4.



DG-08952

Figure 4.4 Break file name

The first six digits following the ? represent the PID (process identifier) of the process within which the terminating program is running. The seventh digit represents the program execution level (swap level) of the terminating program.

Any existing file of the same name in the system working directory is overwritten.

The current memory image of the terminating program as well as information about the program state, task states, user overlays in use, attached segments, and all open files is written to the break file.

Break files are available for later perusal, but they are not restartable, i.e., they cannot be reexecuted.

Process Environment Information

The environment of a process consists of

the searchlist
the working (current) directory
the process priority
the standard input and output channels (?INCH and ?OUCH)

These parameters are specified in the process definition packet when a process is first activated. Initially, the environment of a new process is that specified by the program issuing the ?PROC call. However, modifications of the environment are possible with system calls.

MP/AOS provides a number of calls for interrogating the system for information about a specific process and about system parameters. Additionally, you can change a process' priority, searchlist, and working directory.

Information About a Process and About the System

The ?EINFO call provides information about a specific process, such as

- current memory allocation
- process running state
- elapsed time
- total CPU time
- amount of I/O performed
- number of characters transferred
- open channel count
- number of active tasks
- whether or not the process is the initial process
- whether or not it is blocked
- the priority of the process.

This information can be obtained about any process for which a process ID is known, including the calling process itself.

The PID option to the ?GNAME call returns the fully-qualified pathname of a specific process when you provide the PID.

The ?SINFO call provides the following information about the system:

revision number
memory
allowable number of concurrent processes
system ID
system disk.

Modifying a Process Environment

Once activated, a process can use various calls to modify its environment:

- the searchlist can be retrieved with ?GLIST;
- the searchlist can be modified by means of ?ALIST which adds a directory name or ?SLIST which sets the searchlist;
- the working directory can be modified with ?DIR;
- the process priority can be raised or lowered with the ?PRI call using the PRC option.

Process Scheduling

Although scheduling is primarily a system function, MP/AOS allows you to control it. A process can be prevented from executing by being either blocked or disabled from scheduling. Two calls, ?BLOCK and ?UNBLOCK, provide blocking. The calls ?ERSCH and ?DRSCH control disabling/enabling of scheduling.

Blocking and Unblocking a Process: ?BLOCK/?UNBLOCK

A process is blocked whenever it is the target of a ?BLOCK call. A process can block itself as well as any other process for which the process ID is known. Once blocked, a process is not rescheduled until *another* process unblocks it by issuing an ?UNBLOCK call.

NOTE: *A blocked process has no means to unblock itself.*

Disable/Enable Scheduling: ?DRSCH/?ERSCH

A process can disable scheduling for all processes other than itself by issuing a ?DRSCH call with the PRC option. In this case, scheduling is reenabled only when the executing process issues an ?ERSCH with the ?PRC option, or a ?BLOCK call directed at itself.

Caution must be exercised by a process when issuing a ?DRSCH call; if all the tasks of that process pend, the system hangs, since it has no way of detecting this condition.

System Call Summary

Table 4.1 lists MP/AOS system calls for process management. Task-specific calls are summarized in Table 8.1.

Call	Function	Options
?BLOCK	Block a process	
?BOOT	Shut down the system	
?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)
?MYID	Get task or Process ID and priority	PRC (return process ID and priority)
?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	

Table 4.1 Process system calls

Memory Management

MP/AOS supports up to 1024K words (2 megabytes) of physical address space, 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.

The typical 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.

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

Each user process has a maximum logical address space of 32K words (65536 bytes). However, the use of dynamic memory segments with the hardware mapping feature enables a process effectively to address all physical memory not used by the system, provided the addressing extends to no more than 32K words 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 responsibility of memory management rests with those user processes sharing system resources at the time.

MP/AOS' dynamic memory arrangement 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 MP/AOS multiprocessing environment. This chapter describes the memory environment provided by MP/AOS and the facilities available for memory management.

The MP/AOS Mapping Capability

With a mapped system, you can address up to 2 megabytes of memory. This is done with the aid of the ECLIPSE address translation hardware and the logical-to-physical address translation functions set up by the supervisor program of the operating system. See Figure 5.1.

The memory pages allocated to a user program are not necessarily contiguous. The map feature allows a different logical-to-physical address computation to be specified for each 1K word of logical memory.

The address translation function which correlates a logical address to the corresponding allocated physical memory address is called an *address map*.

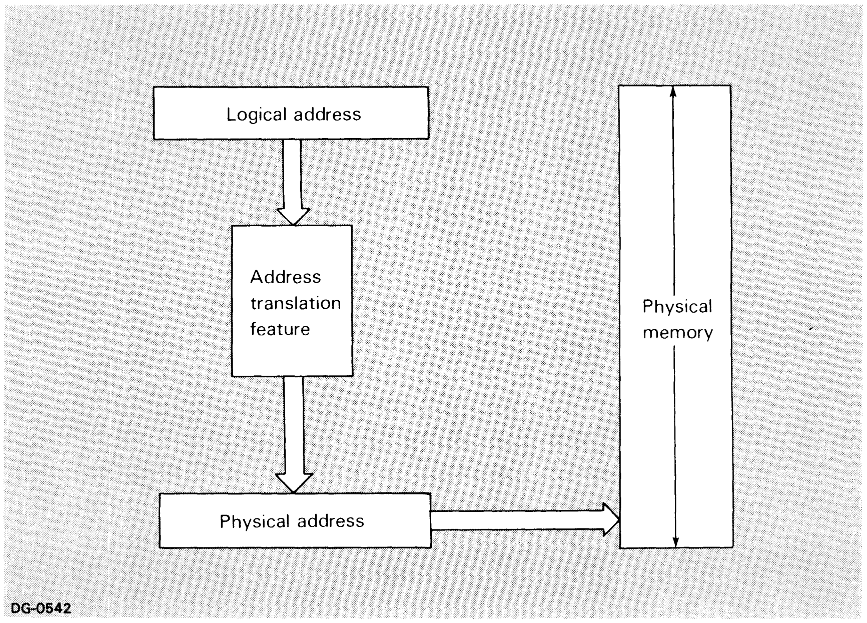


Figure 5.1 Addressing with the MAP feature

Two of these address maps (or four in the case of the S/20 and S/120 processors) 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 BMC (burst multiplexor channel). Data channel and BMC maps can be manipulated by user-written device drivers. (See Chapter 10, "User Device Support.")

In addition to translating addresses, mapping also provides

- validity protection for currently unused portions of the process' logical address space (up to 32K words);
- write protection for certain blocks of allocated physical memory; (Under MP/AOS, shared and overlay memory areas are write protected.)
- indirect protection for the user program; (This prevents the disabling of the system by an *indirection loop,* an indirection chain exceeding 16 levels.)
- I/O protection controlling access to I/O devices. Under MP/AOS, the I/O protection bit and LEF (Load Effective Address) mode are controlled simultaneously: with I/O enabled, LEF mode and the I/O protection bit are disabled (and vice versa). Initially, I/O protection and LEF mode are enabled in user programs, and can be modified with system calls.

Memory Segments

Each *memory segment* consists of a user-specified number of pages (1K block units) of address space. A segment need not be a physically contiguous area of memory. It is a logical entity and can be made up of various available physical pages from different areas of main memory. The system keeps track of segment units, so that, to the user process, a segment can be addressed as a logical area of contiguous locations.

Segments are allocated in multiples of physical pages; therefore, the minimum segment size is one page (1K words). The largest allowable size for a segment is ?MXSP.

Operating System Memory

Since the MP/AOS operating system and user programs occupy different memory areas, there is no danger of a user program overwriting a portion of the system.

In order to optimize system and interrupt performance, the MP/AOS system memory is divided into unmapped and mapped portions. See Figure 5.2.

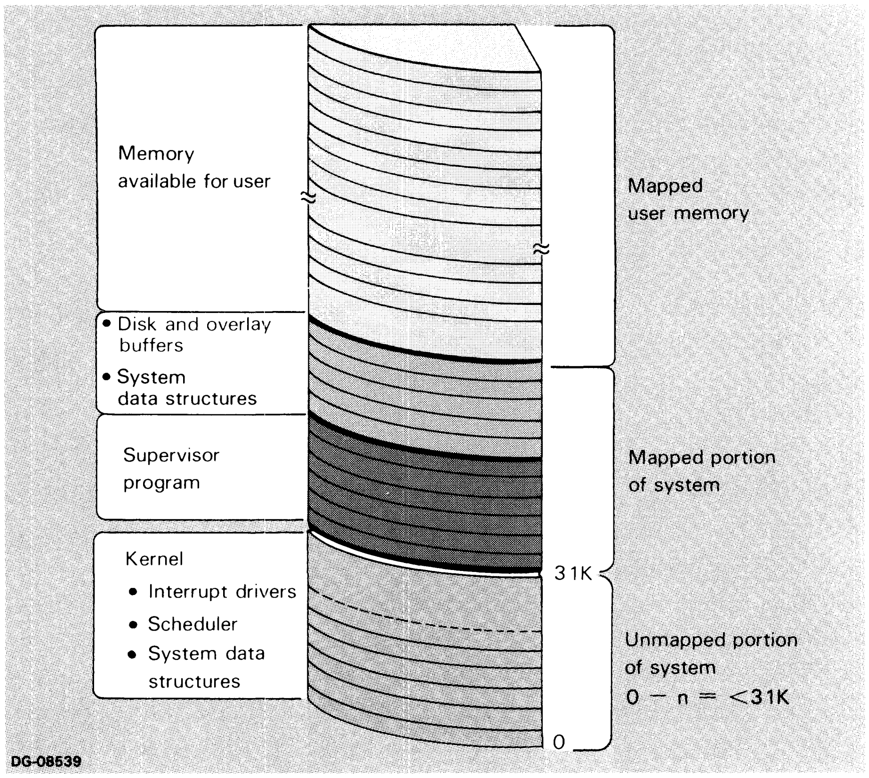


Figure 5.2 System memory configuration

The unmapped portion of the system is the *kernel*. It occupies lower page zero and can extend up to 31K words, 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* and various system data bases and buffers. The supervisor program is overlaid in order to minimize total physical memory requirements.

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

Disk Buffer Area

The preallocated disk and system overlay buffer areas are dynamically mapped. The disk buffer area for disk I/O serves to increase the efficiency of data transfer and to minimize disk access. The *size* of the buffers is predetermined; the *number* of buffers 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, this software-maintained buffer cache is maintained on an LRU (least recently used) basis: the buffer most recently filled is last in line to be flushed to disk when new buffer space is required.

User Process Memory

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 programs running in different processes. Sharing takes places 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 which are only read into memory when the program actually needs them.)

When a process is created (via the ?PROC system call), word ?RMEM in the process definition packet indicates the maximum number of memory pages allocatable to that process, up to 32K words.

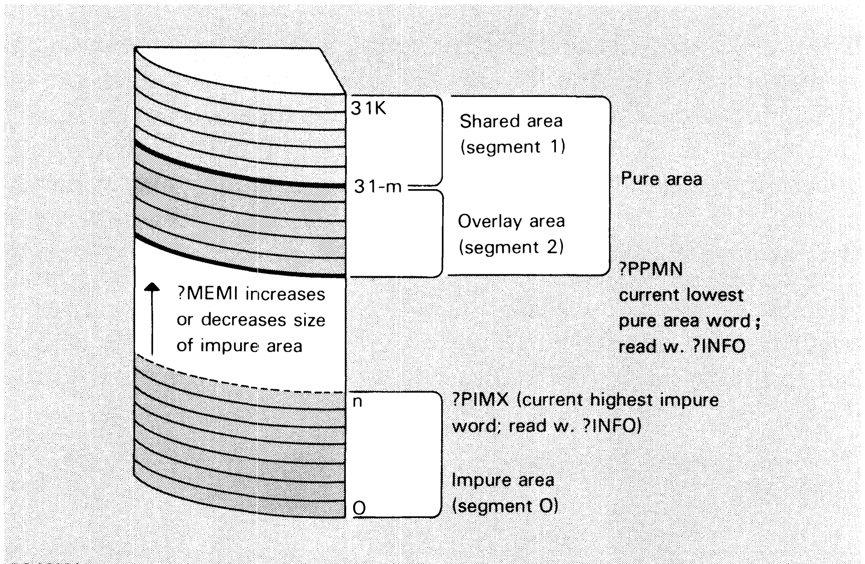
Process memory is initially allocated in up to three memory segments corresponding to the program's impure, shared, and overlay memory needs.

The size of the impure and shared/overlay segments (up to 32K words) is determined by MP/AOS from the program header information set up by the Binder.

The three default segments are automatically mapped into the new process. Each receives a local number 0-2 relative to the process as follows:

Segment 0	Impure area
Segment 1	Shared area
Segment 2	Overlay node area

This allocation is illustrated in Figure 5.3.



DG-08954

Figure 5.3 Organization of user logical address space

The local numbering for these default segments serves a double function:

- It acts as a shorthand reference for each program in dealing with its impure, shared, and overlay memory segments.
- It prevents other programs from referencing these areas, since locally numbered segments cannot be transmitted between processes.

Subsequent programs executing within the same process space are allocated impure, shared, and overlay memory in accordance with the information supplied by the Binder in the .PR image. Note, however, that all such programs are constrained in their total memory size by the value of the ?RMEM word: a program requiring more memory than the total specified by ?RMEM cannot execute.

A program's pure memory area (segments 1 and 2) is fixed in size. In contrast, the impure area (segment 0) is allowed to grow and shrink within the maximum range specified by the user in the ?RMEM word when the process within which the program runs was created.

Modifying the Impure Area, Default Segment 0

The ?MEMI call allows you to request or to relinquish segment 0 (impure memory) as needed, provided that

- sufficient physical memory is available
- the value of ?RMEM (total memory for programs within the process) is not exceeded.

To ascertain the amount of space available between the current impure code boundary and the limit of the impure segment, calculate the difference between the current highest impure address (the ?PIMX value returned in the ?INFO call's packet) and the lowest pure address (the ?PPMN value returned in the ?INFO call's packet).

The difference is the number of words of available addressing space that can be acquired to expand the impure segment, but it is necessary to keep in mind the constraints mentioned at the beginning of this section.

Note that if the program uses neither shared nor overlay memory areas, location ?PPMN is undefined. In that case, calculate the difference between the total pages of memory allocated the process (?RMEM in the process definition packet) and the current highest impure memory (?PIMX).

It is important to remember that ?MEMI specifies memory in *words,* for MP/OS compatibility, and that MP/AOS only allocates memory in one-page increments. Therefore, segment 0's addressing area increases in *full page* allocations, rather than by the specified number of words. For example, when current impure is *n* pages, a request for one additional word to ?MEMI results in the additional allocation of an entire page, although ?MEMI reflects only the addition of one word. Caution must be used because, while machine instructions are not prohibited from accessing remaining page locations beyond the impure boundary, MP/AOS does not allow system call results and/or inputs to access this area.

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 of any size up to ?MXSP pages, causing them to be attached to its address space, and releasing them at will.

Since any number of processes may attach to a segment once it has been created, the efficient passing of data between processes is greatly facilitated. Another useful 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. Since shared memory is just another segment, code sharing becomes automatic.

A user program can also map desired portions of 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 5.4 summarizes extended memory management.

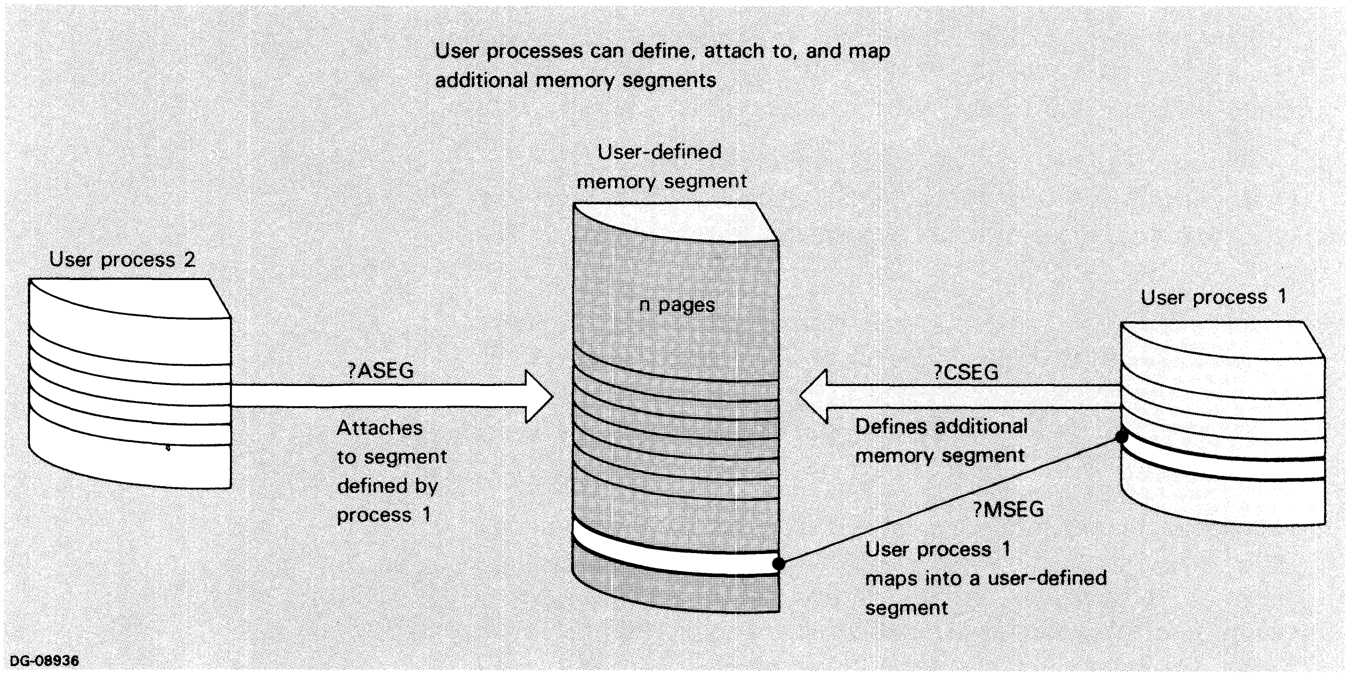

User processes can define, attach to, and map additional memory segments
User-defined memory segment
n pages
User process 2
User process 1
?ASEG
Attaches to segment defined by process 1
?CSEG
Defines additional memory segment
?MSEG
User process 1 maps into a user-defined segment
DG-08936

Figure 5.4 Memory management

Defining Additional Memory Segments

A process can create a segment of any required size up to ?MXSP pages with the ?CSEG call. (Note that at the time of this writing ?MXSP is set to 128.) The total number of memory segments an MP/AOS system will support, including default segments allocated to each user process, is a system generation parameter.

Each user-created segment is allocated a unique global segment number which is returned by the ?CSEG call. The global segment number is important for other programs that wish to attach and map to the same segment.

The new segment is automatically attached to the creating program. It is not, however, mapped in to any user address space.

All pages of a newly-created segment are initially zeroed. Dynamically allocated segments are unswappable resident areas while they are in use.

A program can deallocate a dynamic segment with a ?DSEG call and free up the memory space for other programs, provided no other program is attached to the segment. The termination of a program causes MP/AOS to issue a ?DSEG for every segment attached to the program.

A special extended I/O packet allows segment I/O operations to be performed with user-defined segments regardless of whether or not they are mapped to a given user address space. This facility makes possible direct data transfer between a device and a memory segment. Chapter 9, "Input and Output", discusses this feature in greater detail.

Sharing Memory Segments

Once a segment has been created, several programs executing in different processes can utilize the same memory segment by attaching to it with an explicit ?ASEG call. The global segment number needed for the call can be passed by the segment's creating process via the Interprocess Communication (IPC) facility described in Chapter 7. Attaching to a segment does not map the segment to a process; the ?ASEG call merely increments the segment's use count. The total number of attached segments (including process default segments) that an MP/AOS system will support is specified at system generation time.

This technique allows you to specify a segment for later use, but does not require that you map to it. If the segment-creating process terminates before you attach to its segment(s), MP/AOS deallocates the segment(s). To ensure that a needed segment is available, position the attach calls early in the program. The maximum number of attachable segments for a process is specified by the ?RMAS value of the ?PROC call's process definition packet.

Mapping Memory Segments Within a Process

The ?MSEG system call maps portions of memory segments into user logical address space as illustrated by Figure 5.5.

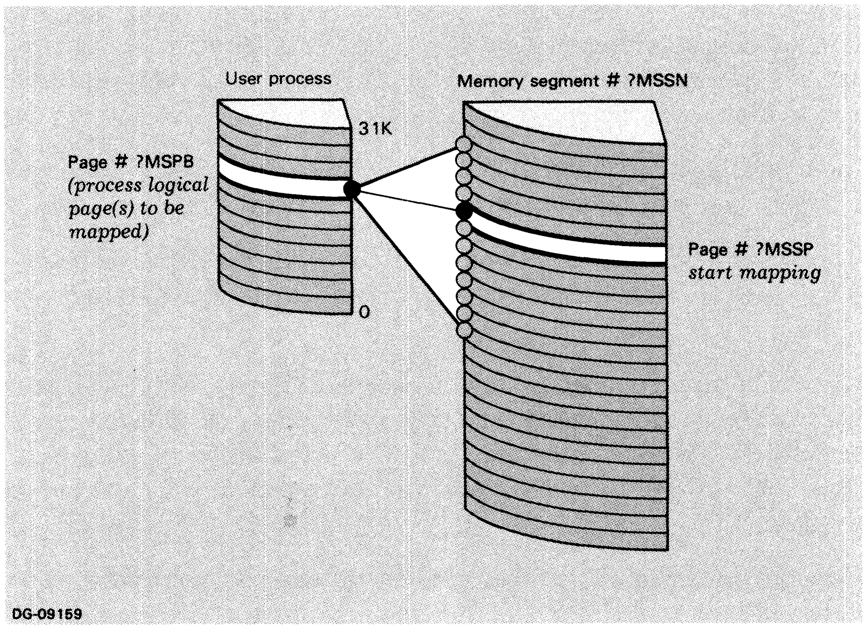


Figure 5.5 Mapping to a segment

Mapping is done for specific logical pages of a segment.

The call requires the segment number, the segment's starting page number and the total page count (or entire segment indicator) of the segment being mapped, along with the process logical page number to which mapping is being done.

Any number of consecutive pages can be mapped with a single call.

The ?MSEG call has a write-protect (WP) option that traps any attempt to modify the protected mapped portion of the segment.

The mapping function includes the specified segment pages in the map for the calling process, making the new memory space addressable for that process. The former contents of those portions of user logical address space which are mapped to a new segment become inaccessible unless remapped.

Dynamically mapped segments, unlike the impure default segment, are not swapped out and in by the ?EXEC and ?RETURN calls.

A second call is not necessary to unmap a segment. A subsequent ?MSEG call overwriting the same physical area serves to unmap all physical pages of the segment mapped to that same area. (Managing the mapping of a number of segment areas to a process is the responsibility of the calling process.)

System Call/Library Routine Summary

MP/AOS system calls for memory management are summarized in Table 5.1.

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)
?OVLOD	Library routine: Load an overlay	
?OVREL	Library routine: Release an overlay	

Table 5.1 Memory calls summary

Process Debugging and Runtime Performance Monitoring

This chapter describes the MP/AOS facilities available to users who wish to write their own debugger and runtime performance monitoring (*histogramming*) programs instead of using the programs available under MP/AOS. The chapter is divided into two parts: the first part describes MP/AOS debugging facilities; the second summarizes MP/AOS *histogramming* facilities. The available system calls are described in detail in Chapter 12, "Dictionary of System Calls and Library Routines".

Process Debugging

MP/AOS allows you to debug a program without having to bind a debugger with it. FLIT, the MP/AOS process debugger, is an independent program that runs concurrently with the executing program within a different process space and is activated whenever the system blocks the executing program. Program blocking occurs anytime MP/AOS detects either a user-specified program break (*breakpoint*) or an automatic breakpoint condition. Whenever the system detects such an event, it generates a signal that is intercepted by the debugger.

FLIT is an interactive system utility. Debugging a multitasked program in a simulated real-time environment allows you to examine and/or modify an actual "snapshot" of the executing program and provides current program status information. FLIT's full capabilities are described in detail in MP/AOS *Debugging and Performance Monitoring Utilities* (DGC No. 069-400205).

The MP/AOS support for users desiring to write their own debugger programs includes the following system calls:

- an Await Signal (?WSIG) call to intercept signals from the executing program, pass control to the debugger, and resume program execution;
- five 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.

Invoking the Debugger

The debugger program invokes the program to be debugged by issuing a ?PROC system call with the debug (DB) option. MP/AOS sets up the program for execution within the target process and passes control to the debugger before any user instructions are executed. At this point the debugger can investigate the program status and set breakpoints.

The debugger program and the target program execute in separate processes. Both processes reside in main memory during debugging.

To unblock the program and resume its execution, the debugger issues an Await Signal (?WSIG) call. ?WSIG sets the signal mask specifying the signals of interest to the debugger process.

The program being debugged executes and the debugger waits while the operating system screens the program for these events.

Whenever MP/AOS detects any of the signals specified in ?WSIG's signal mask, ?WSIG returns the signal number to the debugger. MP/AOS blocks the user program and halts execution, and the debugger process then takes control; debugger system calls can then be issued to inspect and modify the target program's memory location contents.

When MP/AOS blocks the process, the status of the interrupted user program is saved so that the debugger process can restart the program's execution at the interrupted instruction. ?WSIG returns to the debugger program the value of the program counter as well as the task ID of the task currently active in the program being debugged.

Signals

The events that cause MP/AOS to block the program and generate a signal intercepted by the debugger can be explicit user-specified program breakpoints or events indicating error conditions, potential exception conditions, or system calls.

There are four classes of events and associated signals. Each signal and signal class is designated by a mnemonic corresponding to a number. The mnemonics and their numeric values are listed in OPARU.SR, the user parameter file. See Appendix E.

The four signal classes are:

user catchable signals
signals causing program abortion
breakpoint signals
system calls

User Catchable Signals

Table 6.1 lists the signals in this class. The user can provide exception handling routines for each of the signals in this class and bind these routines with the program. In the absence of user-provided routines, MP/AOS force binds with the user program a routine to direct control to the system whenever a signal in this class is encountered.

Class	**?SGUC**
?SGSO	Stack overflow
?SGFP	Floating exception
?SGCI	Commercial exception

Table 6.1 User catchable signals

Abort Signals

Table 6.2 lists the signals in this class.

Class	?SGAS
?SGJO	User JMP 0
?SGVT	Validity trap
?SGWP	Write protection trap
?SGIO	I/O protection trap
?SGIT	Indirection protection trap
?SGCB	^C^B console interrupt by user
?SGCE	^C^E console interrupt by user
?SGRI	microECLIPSE reserved instruction trap
?SGRT	Program termination (?RETURN, all tasks ended)
?SGRB	Program termination (?RETURN BK, all tasks ended, checkpointed)
?SGKL	Program termination (?KILL, all tasks ended)
?SGKB	Program termination (?KILL BK, all tasks ended, checkpointed)

Table 6.2 Abort signals

System Calls

Table 6.3 indicates the signals in this class. Two system calls, namely, ?IXIT and ?IUNPEND, do not generate signals, because these calls are issued from interrupt level only.

When the program halts in response to a system call signal, the halt takes place *prior* to the execution of that system call. To allow the system call to execute without being trapped once more when the program resumes, a ?WSIG call with the Resume (RE) option must be issued. This option causes MP/AOS to process the system call last detected without generating a signal to the debugger; the program resumes normal execution until the occurrence of the next signal or system call.

Class	?SGCL
?SGCL	System call SVC*

Table 6.3 System calls

**SYC, sometimes named SCL or SVC is a hardware instruction that pushes a return block on the caller's user stack and transfers control to a kernel routine which activates either a system or breakpoint handling routine as appropriate. The SYC instruction is discussed in the Principles of Operation manual appropriate to your processor.*

Breakpoint Signals

Table 6.4 lists the signals in this class. These signals include all the breakpoints you set with the debugger program, the manual console interrupt signal, an overlay loaded signal, and the issuing of an ?EXEC call. (When the program being debugged issues an ?EXEC call, the debugger program debugs the new program).

Class	?SGBP
?SGUS	Unknown SVC*
?SGBK	Breakpoint SVC* (SVC 0,1)
?SGB2	Secondary breakpoint SVC* (SVC 1,0)
?SGCD	^C^D (enter debugger)
?SGEX	?EXEC issued
?SGOL	Overlay loaded

Table 6.4 Breakpoint signals

**See footnote fo Table 6.3.*

Debugger Operations

Five system calls permit the debugger program to examine or modify program status. These calls, discussed in the following sections,

retrieve all task ID's in the program (?GIDS),
read or write task state (?RDST and ?WRST),
display or modify memory (?RDMEM and ?WRMEM).

Retrieving Task Identifiers in a Program

The ?GIDS system call allows you to retrieve the task identification number of all tasks in the program being debugged. With a specific task identification number, you can then use the ?RDST and ?WRST calls to view and modify the status of any task in the target program.

Examining and/or Modifying a Task State

The ?RDST call returns a packet indicating all task status information (task status word, priority, pend key, timeout data, and the number of the last overlay loaded). This call returns, in addition, the contents of all four accumulators (AC0 through AC3), along with the program counter, the carry bit, and the contents of the stack and frame pointers. The value of the Unique Storage Position (USP), the four floating-point register contents and all floating-point status information is also returned.

Some task status values such as task priority are protected from modification and available as read-only information. Other values can be modified by issuing the ?WRST system call and specifying the new values in the appropriate offsets of a Write Task State Packet.

Modifiable task state values include

the accumulator and floating-point register contents,
program counter,
carry bit,
frame and stack pointer values,
stack limit,
Unique Storage Position.

Examining and Modifying Memory Locations

To examine a given memory location in the target program, issue the ?RDMEM system call specifying the desired starting address as well as the number of words to read. To modify the contents of a given memory location, issue ?WRMEM, specifying the starting address and the number of words to be modified.

Resuming Program Execution

Program restart is provided by another ?WSIG call causing the program to resume execution at the place at which it was interrupted (i.e., the current value of the Program Counter). Since ?WSIG allows the signal mask to be set, the event classes generating signals can always be modified prior to program resumption.

Runtime Performance Monitoring

MP/AOS provides two system calls and a utility program enabling the user to monitor program execution and determine bottlenecks.

The two system calls, ?EHIS and ?DHIS, allow continual sampling of processes to determine the proportion of execution time spent in each region of the address space. The frequency distribution generated by these system calls is referred to as a *histogram*. ?EHIS (Enable Histogram) gathers the data and places it in temporary storage locations; ?DHIS (Disable Histogram) causes the data-gathering to stop and makes a summary of the data accessible.

The utility program PROFILE is built around the system calls and provides an easy-to-use interface and an easy-to-read report. The maximum number of concurrent histogram operations supported by an MP/AOS system is user specified when the system is generated.

The region monitored may be the entire address space or a portion of it, as specified by the user. The user also selects the number of times the process is monitored per second and the interval size used to create the histogram. Considerations in selecting frequency, range, and interval size are explained in the system call dictionary.

In most instances, the PROFILE program can be used in place of the ?EHIS and ?DHIS calls. This program automatically builds a histogram and reports it in an easy-to-read format. The user can vary the data gathering and reporting parameters, and can use the program interactively or with a command file.

If you use the system calls instead of PROFILE to generate a data buffer, you can still use PROFILE to generate a printed report. To do so, incorporate the data buffer in a data file conforming to the format described in the PROFILE documentation in Appendix D of *MP/AOS Debugger and Performance Monitoring Utilities* (DGC No. 069-400205).

The PROFILE program is described in the same manual.

System Call Summary

MP/AOS debugger and histogrammer calls are summarized in Table 6.5.

Mnemonic	Function	Option
?DHIS	Disable histogramming	
?EHIS	Histogram a process	
?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 6.5 Debugger, histogrammer system calls

Interprocess Communication

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

The inclusion of the Interprocess Communication (IPC) facility in an MP/AOS system must be specified when the system is generated.

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

The Interprocess Communication facility is also useful for synchronizing process activity, passing data between processes, or notifying other processes of execution results.

Initially, a customer-server relationship must be established between two processes; that is, one process must be designated as a *server,* and the other as a *customer,* regardless of the desired relationship between the processes. Once the initial contact has been established, the two parties can communicate directly with each other.

IPC calls allow either party to send and receive messages, 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 the connection between itself and all of its customers, and to obtain a list of all customers who have broken connection with it.

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

Tasks issuing IPC calls can continue activity without delay by using the *nonpended* option: this causes the system to create separate tasks to perform the IPC action. These system-created tasks are suspended until completion of the IPC action or expiration of the specified time limit, while the tasks issuing the IPC call continue normally.

Customer/Server Relationship

The customer-server relationship is the mechanism for establishing a connection between two processes. The server process declares itself as a server by means of the ?DCLR system call.

Once the process is a declared server, any number of customer processes can establish a connection with it. Customer processes need not declare themselves as such. The customer merely looks up a given server process by name (?LKUP); if the call is successful, a connection is established between the two parties, allowing them to begin communicating.

A process can fill server functions for a number of customer processes. Figure 7.1 shows a server process with connections to three customer processes; as the figure illustrates, the server process can also act as its own customer.

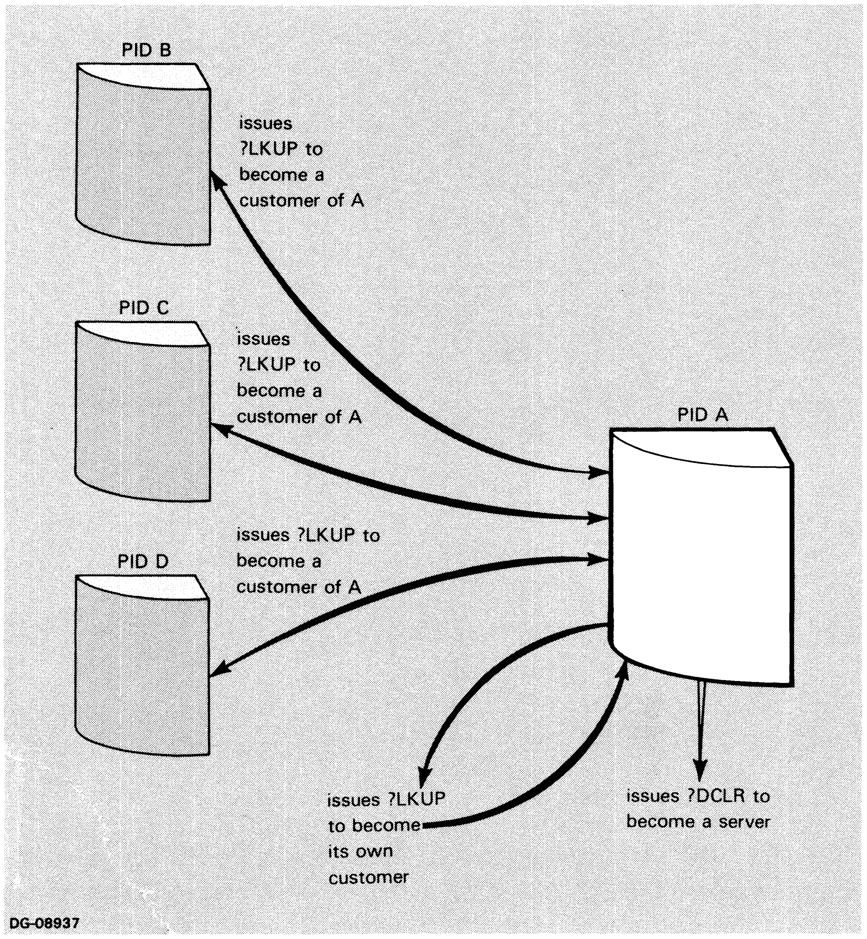
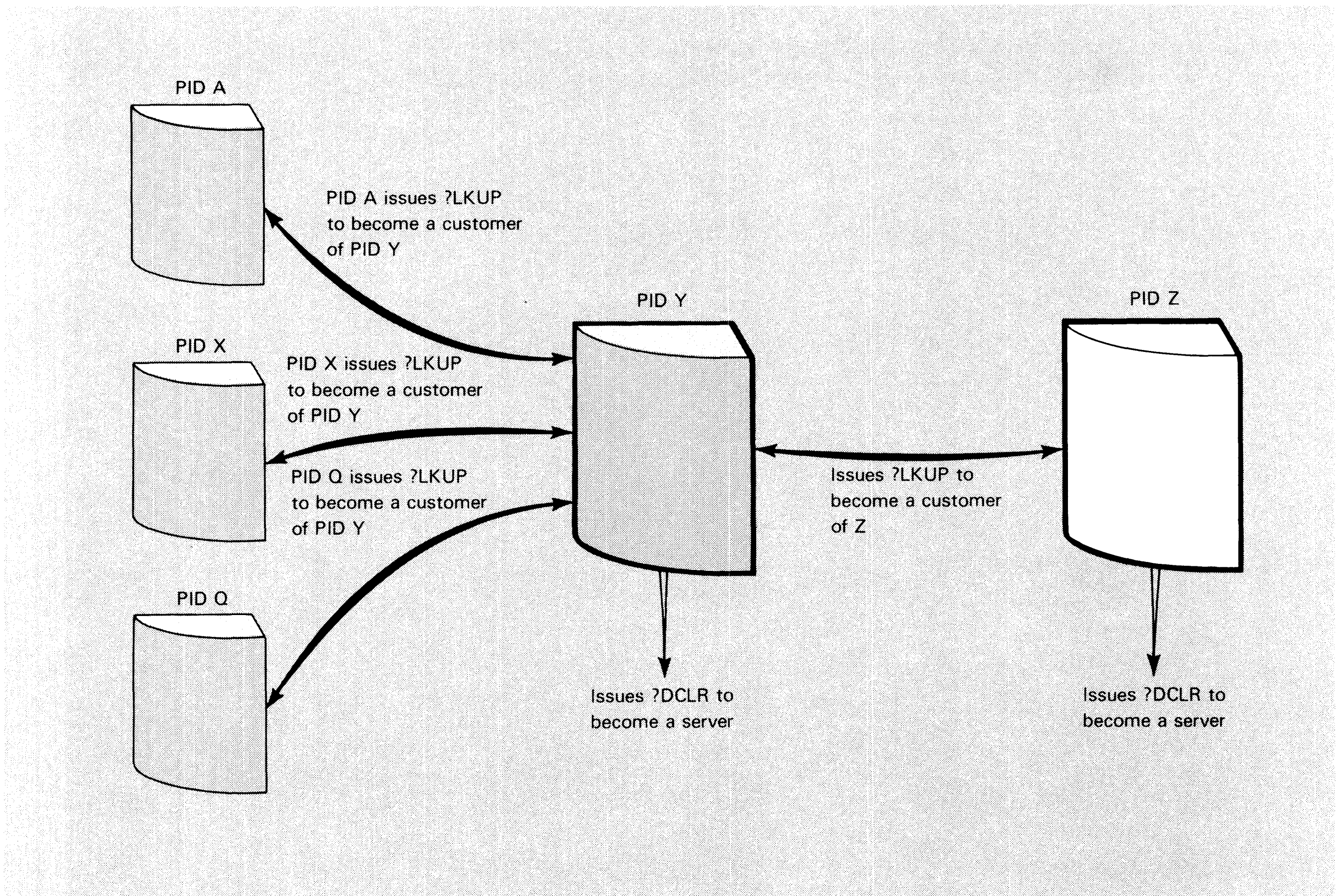

PID B
issues ?LKUP to become a customer of A
PID C
issues ?LKUP to become a customer of A
PID A
issues ?LKUP to become a customer of A
PID D
issues ?LKUP to become its own customer
issues ?DCLR to become a server
DG-08937

Figure 7.1 Sample ‹'ustomer/server configuration

It is possible for a process to act as a server for some processes and as a customer of others. Figure 7.2 shows a *multilevel connection,* where process Y is the server of processes A, X, and Q, and a customer of process Z. Multilevel connections let you set up intermediate servers for some functions, and one or more superior servers for other functions.



DG-08938

Figure 7.2 Sample multilevel customer/server configuration

IPC calls

The role of the originating process determines which IPC system calls may be used. Note that many calls are common to both server and customer processes, among them, calls for sending and receiving messages, for breaking off the association with another process, and for requesting the process ID of the other party. Table 7.1 illustrates.

Server only	Customer only	Server or customer
?DCLR	?LKUP	?SEND
?RCVA		?RCV
?RMVE		?CLEAR
?PURGE		?SD.R
?OBITS		?TPORT

Table 7.1 IPC calls and processes issuing them

Many of these calls require a packet of specifications. The packet specifies such items as the address and length of the message and/or message buffer and other information pertinent to the connection. Packet formats and offset words are discussed below.

IPC ports

MP/AOS sends IPC messages between *ports,* which are full-duplex communication paths. A customer process receives (via ?LKUP) a port number for communicating with a given server. When the server process receives a message from that customer, the receive any (?RCVA) call returns a port number identifying that customer. Customer and server subsequently use the port number each has been allocated for communicating with the other.

Using IPC Calls

The following steps describe a typical IPC sequence:

- Server process specifies a name and declares itself a server with the ?DCLR call. ?DCLR returns a server number for self-identification in server-only calls.
- A process with a message for this server looks up the server name with the ?LKUP call and is allocated a port number for conversing with this server. A connection is now established between the two parties.
- The customer process initiates the communication by sending a message (?SEND) to the server port it has looked up.

  [Alternately, the customer can issue a send/receive call (?SD.R).]
- The server process issues ?RCVA, enabling it to receive messages from *any* customer process. A port number identifying the sending process is returned along with the received message.

  [Henceforth, the two parties can, if they wish, conduct a "private conversation" (server issues ?RCV to pend on messages from a specific customer port number).]
- The server process replies to customer's message via ?SEND, specifying the port number allocated to the customer and returned to server in ?RCVA.
- The customer process issues a ?RCV to receive the server's message (unless a send/receive call, ?SD.R had been previously issued).

  The two parties repeat the sending and receiving process at will.
- The customer or server issues ?CLEAR to terminate the exchange.
- The server issues ?RMVE to disassociate itself from a name. Server issues ?PURGE to break the connection between itself and all other processes.

The send and receive calls need not be sequential. If there is no outstanding message for a receiver, MP/AOS either suspends the receiving task for the specified time interval until the ?SEND is issued or returns an error.

Server Declaration and Customer Connection

As stated, a process must explicitly declare itself as a server process via the ?DCLR call. In ?DCLR a process declares a specific name by which it is known to a customer process. A server can issue several ?DCLR calls specifying several server names; but a given server name must be unique to only one process at a time. ?DCLR returns a *server number*. The server process uses this server number as its identification when issuing server-only calls such as ?RCVA, ?RMVE, ?PURGE, and ?OBITS.

Server Table

The system maintains a system table of server descriptions called the *server table* with an entry for each server. After a process has successfully issued a ?DCLR call, the system adds an entry for it in the server table. The size of the server table is a system generation parameter.

Communication can now be established with this server process. To do so, a customer process must "look up" the server; that is receive a port number for communicating with the server.

The customer looks up a server by specifying the server name as an input to the ?LKUP system call. ?LKUP returns a *port number* which the customer can use to communicate with that server process. (A customer process can issue more than one ?LKUP on the same server name and receive more than one port number for conversing with it.)

Once the customer process has specified a valid server name in ?LKUP and has been allocated a port number to communicate with that server process, a *connection* has been established between the two parties.

Connection Table

The system maintains a table of connection descriptors called the *connection table*. When a customer establishes a connection with a declared server, the system adds an entry to the connection table. (Each successful ?LKUP generates one connection table entry.)

The table size, i.e., the maximum number of permissible connections, is a system generation parameter.

Sending and Receiving Messages

The server and customer processes use the same system call (?SEND) to send messages to each other. Initially, the server process has no means of identifying its customers: it does not know when and by whom it has been looked up, until it receives a customer's first communication. Hence the customer process must initiate the conversation by issuing the first ?SEND.

The first customer message is received by the server via a general ?RCVA call which accepts messages from any process. Successful receipt of the first message identifies the sending process to the server by the customer port number returned in ?RCVA. Once the customer is identified, the two parties can communicate over their respective server and customer port numbers rather than over the more general server receive port. That is to say, the server process can issue ?RCV calls to await messages from specific customer port numbers, thus conducting symmetric conversations with its customers.

To prevent ?RCVA from intercepting messages for which a specific ?RCV is posted, ?RCV calls have precedence over ?RCVA calls. If a server has a task pended on a ?RCV on a specific port and another task pended on a ?RCVA, a message from the specified sender port is received by ?RCV and not by ?RCVA. This is true regardless of which call was issued first.

?SEND and ?RCV calls are serviced on a first-in, first-out basis. The task first issuing the call has it transmitted; this is true for tasks within a program, as well as for tasks within different programs.

Send and Receive Packets

The send and receive packets are of the same type (?SRP) and length (?IPCLN).

The send/receive (?SD.R) packet is identical to the ?SEND packet, but with an extension for receive information, since ?SD.R performs both send and receive functions. The ?SD.R packet (type ?S_RP) consists of ?ISRLN words. Table 7.2 shows the structures of the IPC packets.

Offset	Contents	
?ITLM	Time limit (seconds) to wait	
?IMAD	Message byte address	Type ?SRP only
?IMLN	Message byte length	
?ISAD	Message byte address	Type ?S_RP only
?ISDL	Message byte length	
?IRAD	Receive area byte address	
?IRLN	Receive area byte length	

Table 7.2 IPC send, receive, and send/receive packets

Offset ?IMAD in the send packet points to the start of the message in the sender's logical address space, and offset ?IMLN contains the length of the message. Within the receive packet, these same offsets describe the starting byte address of the receive buffer and its size respectively.

The receive buffer address can be in any part of the user address space, as well as within a user-defined memory segment currently mapped to the user address space, provided that area is not write protected.

If insufficient space is available to transfer the message when a ?SEND or ?SD portion of a ?SD.R request has found the target process' receive request, an error is propagated to the receiver only. AC1 returns the length of the entire message to the receiver, enabling it to modify the receive buffer size and reissue the receive call.

Within the time limit it has specified, the sender meanwhile continues waiting for the receiver as if it had never found the receiver process.

Terminating a Customer/Server Relationship

The system breaks a connection between two processes under any of the following circumstances:

termination of either party (program/process termination)
?EXEC call issued by either party
?CLEAR call issued by either party
?PURGE call issued by server

When the system detects a broken connection, it sets a flag bit in the appropriate connection table entry. However, the table entry is not cleared until disconnects have been received from both the customer and the server: if, for example, a customer terminated or issued a ?CLEAR to break its connection with the server, the PIDs of both processes remain in the connection table until the server process issues either a ?CLEAR or a ?PURGE.

The user should issue disconnects from both processes as soon as a connection has served its purpose. This keeps the number of connection table entries within the maximum range.

Program/Process Termination

All connections are broken and all server names are removed when a program with outstanding connections or server declarations

terminates
"pops" a level (?RETURN)
is swapped out (?EXEC).

Tasks of other programs pended on IPCs to the terminated or ?EXECuting program take error returns. Similarly, tasks pended on ?SEND, ?RCV, ?RCVA, ?SD.R, and ?OBITS calls within the ?EXECuting program take error returns.

Issue of ?CLEAR by Either Party

At any time, either party can terminate the association existing between it and the other party via a specific port number by issuing a ?CLEAR call. This causes any outstanding send or receive messages to and from the caller to terminate in error.

NOTE: *A connection is not broken unless the appropriate calls have been issued by both parties.*

Issue of ?PURGE by Server

To break a *single* connection to a customer, a server issues ?CLEAR. To terminate its function as a server, the process must disassociate itself from its server name so that no new customers can look it up. It must also break the association between itself and *all* of its current customers. ?RMVE removes the server process from the association with the name used in the ?DCLR call. ?PURGE takes the server port number as an input and breaks all connections which may exist between this process and any other processes communicating with it. Any outstanding calls to or from the issuing process terminate in error.

Discovering Broken Connections

A process can discover in one of two ways that a connection with another process has been broken:

- a ***No connection exists*** error on send/receive calls
- a ?OBITS call.

Servers awaiting customer messages through ?RCVA (receive any) cannot take the ***No connection exists*** error since they are not awaiting a message from a specific process. If a customer process breaks its connection to that server (?CLEAR call or program termination), it is difficult for a server process to detect the broken connection.

?OBITS allows a server to determine any broken connections before issuing a ?RCVA. The call returns a list of port numbers representing connections broken by the customer.

Once a connection is broken, ?OBITS repeatedly returns the port number of that connection until the server issues a ?CLEAR, or a ?PURGE call.

Appendix K contains listings of two sample IPC programs.

System Call Summary

IPC calls are summarized in Table 7.3.

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 7.3 Interprocess communication calls summary

Multitasking

Multitasking greatly simplifies certain types of programs, notably those which must perform a number of operations in parallel. The system allows you to divide a program into a number of subprograms called *tasks*.

Multitasking is similar to *multiprogramming,* or timesharing, in that multiple control paths are established. However, all tasks are part of a single program, so they must share memory, I/O channels and other system resources.

An example of a multitasking program is a multi-user editing system that supports several people working at consoles. Under the MP/AOS system, you simply assign a separate task to each user, and the system takes charge, deciding which user to service.

Managing Tasks

The total number of tasks supported by an MP/AOS system is a system generation parameter. Include in this number one system task created for each user process, up to the maximum number of processes specified during system generation.

You must also specify the maximum number of tasks your program(s) will require so as to enable the system to allocate memory for task control information. The limit on the number of tasks that programs executing within a given process can use is set when you issue a ?PROC system call to create that process (word ?RMTC in the process definition packet).

The system-created task for the new process need not be counted as part of the maximum number of tasks you specify for that process. This number should, however, include provision for system tasks created as a result of *nonpended* system calls issued by programs that execute within the process. See Chapter 9 for a discussion of nonpended calls.

At run time, you *create* tasks with the ?CTASK system call. Creating a task is similar to calling a subroutine, but the calling routine continues to run: it does not wait for the called routine to exit. You have the capability to control the contents of AC0, AC1, and AC2 in the created task. AC3 is set to the address of a routine to which the task should jump when it finishes running. Because of this accumulator handling, a task may be written to use the SAVE and RTN instructions just like a subroutine.

When you create a task, the system assigns it a *task identifier* (TID), a 16-bit number used with system calls to reference the task. A task can retrieve its own identifier with the ?MYID call.

The task identifier (TID) is distinct from the process identifier (PID), though both identifiers may be retrieved with ?MYID.

Tasks are deleted (*killed*) when they jump to the address in AC3. You can also kill a task at any time with the ?KTASK call. If you have specified a *kill post-processing routine* for the task, it will be executed at this time. This routine can perform such functions as deallocating memory used by the task. When the routine is entered, AC2 will contain the identifier of the task being killed, and AC3 will contain a return address to which the routine will jump when it finishes executing.

NOTE: *A task kill post-processing routine may* not *execute system calls.*

When you create a task, specify a routine to be called in case the task causes a stack overflow. Before calling this routine the hardware pushes five words onto the stack consisting of

- the accumulators AC0 through AC3
- a word containing the carry in bit 0, and the contents of the program counter (where the overflow occurred) in bits 1-15.

Since the system's handling of a stack overflow involves pushing more words onto the stack, ensure that your stack is at least actually five words larger than the size you specify in the stack limit word. Otherwise, part of your program code may be destroyed during the handling of the overflow. You should also allow stack space that may be needed by the overflow handling routine itself.

If you do not specify an overflow handling routine, any stack error will kill the task.

Stack overflow handling routines return using the POPB (Pop Block) instruction rather than RTN.

Parallel Call Errors

A conflict may arise in a multitasked program if one task executes an ?EXEC or ?RETURN while another task has a system call in progress. A similar situation may occur, even in a single-task program, if you interrupt the program from the console. In these cases, any outstanding calls will be aborted, and they will return an error with code *ERPCA (Parallel Call Abort Error)*.

Task Priority

Tasks, like processes, are scheduled by *priority*, but task priority is subordinate to process priority.

A task priority is designated by a number between 0 and 255; lower numbers represent higher priorities.

You specify a task's priority when you create the task with the ?CTASK system call. A default priority of 177_8 (127_{10}) is assigned by the system to a program's initial task. A task can modify its own task priority, as well as the priority of other tasks within the current program with the ?PRI call.

In a multiprocessing environment, task priority is subordinate to process priority: the tasks within the highest priority *process* are executed according to their relative priority. Tasks within lower priority processes are ignored, regardless of their priority.

Scheduling

At times you will need to suspend multitasking activity; for instance, you may need to read and modify a critical memory location without having some other task modify the same location at the same time. Two system calls support this activity: ?DRSCH and ?ERSCH.

?DRSCH disables the task scheduler and ensures that no task runs except the one that executed the ?DRSCH. When the task completes the critical activity, it re-enables the scheduler with the ?ERSCH call. You can also use ?DRSCH to determine whether or not multitasking is currently enabled, as explained in Chapter 12, "Dictionary of System Calls and Library Routines".

Tasks can suspend and enable multitasking and synchronize their activities within their current program.

Intertask Communication

Tasks within the current program are able to control each other's actions. The system permits synchronization of tasks' activities through the ?PEND and ?UNPEND calls. When a task executes a ?PEND, it is suspended until a particular event occurs. The event is specified by a 16-bit *event number*. This number must be used by another task in a ?UNPEND call to unpend the pended task. ?UNPEND can also unpend a particular task by specifying its task identifier.

Event numbers must be between zero and the value of the mnemonic ?EVMAX. Values between zero and mnemonic ?EVMIN are reserved for system-defined events, which you may specify in a ?PEND call but not in a ?UNPEND call. Values between ?EVMIN and ?EVMAX, inclusive, may be used for either ?PEND or ?UNPEND. ?UNPEND also allows you to pass a one-word message to AC0 of the unpended task.

When you unpend a task, it may take either the normal or error return from its ?PEND call. If it takes an error return, the unpended task should then examine the contents of AC0 to determine the error cause. Be sure that the value of the message word is not the same as one of the ?PEND error codes; otherwise, a task that takes an error return will be unable to determine the cause of the error.

Console Interrupt Tasks

To interrupt a program, users may type a CTRL-C CTRL-A sequence on the console keyboard. To receive this interrupt, your program must create a task pending on an event number equal to ?EVCH plus the channel number of the console keyboard. If the user types CTRL-C CTRL-A, the task will be unpended and free to perform such actions as accepting a command from the user or terminating the program.

System Call/Library Routine Summary

MP/AOS task management calls and library routines are summarized in Table 8.1.

Mnemonic	Function	Option
?CTASK	Create a task	AW (await TCB if none available)
?DELAY	Library routine: delay execution of a task	
?DRSCH	Disable task or process rescheduling	CK (take error return if multitasking already disabled) PRC (act on process)
?ERSCH	Enable task or process rescheduling	PRC (act on process)
?INFO	Get program information	PID (return process information)
?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)
?UNPEND	Resume execution of a task	BD (unpend all tasks) ER (unpend at error return) ID (unpend on task ID, not event code)

Table 8.1 Multitasking call and routine summary

Input and Output

Overview

All data transfers between the user program and a device or file take place via an *I/O channel.* Under MP/AOS, the user program controls the allocation and release of I/O channels.

When a channel is opened to a file, a *file pointer* indicates byte position in the file. The positioning of the file pointer is user controlled to permit random access to any byte in the file.

MP/AOS generally buffers data transfers through a software maintained buffer cache. Buffering is bypassed when entire disk blocks are transferred. For cases in which it is important to keep the user file updated between short transfers (e.g., for the creation of checkpoint records), the Flush option on ?WRITE is provided. This causes all file system data associated with the channel to be written to disk before the ?WRITE call takes a return to the user program.

MP/AOS provides *nonpended I/O,* allowing a task to continue processing overlapped with that task's I/O. The system allows the program to be notified when any of its nonpended calls (tasks) has completed execution.

Options on the ?READ and ?WRITE system calls provide the user with several techniques of data transfer, namely, *dynamic* and *data sensitive* I/O. A packet form permits *extended* I/O to segments not currently mapped to the user's address space.

I/O devices are divided into those with a block structure such as disks and magnetic tape, and those without a block structure, i.e., character devices such as consoles and line printers. Magnetic tape is supported as part of the MP/AOS library.

User calls introduce disk devices to, and release them from, the system. The system performs consistency checks on disks. Disks can be MP/AOS formatted by means of the DINIT utility. MP/AOS disk structure is identical to that used by MP/OS, and disk media are interchangeable between these two systems.

Terminals are handled as two devices, namely keyboards for input and CRT consoles or printers for output. Console and line printer characteristics are user modifiable and offer numerous facilities without programming intervention. Predefined control characters and control sequences are supported. Data channel line printers are supported with a subroutine library and a spooler process.

I/O Channels

All data transfers between a user program and a device or file take place via an I/O *channel*. An I/O channel (not to be confused with a *data channel*), is a system-defined data path.

To use an I/O channel, you must *open* it; i.e., you must connect it to some device or file. Your program does this with the ?OPEN system call. You specify the maximum number of I/O channels that can be opened system-wide at any one time when generating your MP/AOS system. Similarly, you specify the maximum number of I/O channels a program can have open at any one time when you create the process within which the program executes (packet word ?RMCH in the ?PROC system call). Up to 512 I/O channels can be stipulated for all processes in the system.

When you finish using an I/O channel, you can release (*close*) it with the ?CLOSE system call. The ?EINFO system call returns information on the number of open channels in your program, and the ?INFO system call returns the status of the first sixteen (16) I/O channels in your program.

An important feature of the MP/AOS system is its ability to pass I/O channels between programs: when a program performs an ?EXEC, the states of any active I/O channels are passed to the new program. Thus the new program can perform input and output on these channels without reopening them. I/O channels can be passed regardless of whether the program performs the ?EXEC with the chain or the swap option. No channels are passed between concurrent processes created via the ?PROC system call.

The passing of I/O channels is a useful form of communication; it can also be potentially hazardous if the new program does not expect to find any open channels. It is therefore a useful precaution to start all programs by closing any unneeded I/O channels. MP/AOS provides two mechanisms for doing this:

- The CL option of ?EXEC closes all open I/O channels on behalf of the new program, with the exception of the standard input and output channels. All channels are restored after the EXECuted program terminates.
- The ?RESET call allows the new program to close one or more of the first sixteen I/O channels by specifying them in a 16-bit mask. (Channels beyond the sixteenth must be individually released via the ?CLOSE call.)

When a program performs a ?RETURN, the parent program resumes execution with the same I/O status it had when it performed the ?EXEC.

Standard Input and Output Channels

The MP/AOS CLI always opens two channels for console I/O and always passes these two channels to other programs, since almost all programs use them. The CLI always closes all other channels before calling any program.

The standard *input* channel has the mnemonic ?INCH; in the initial process, ?INCH is opened to device @TTI. The standard *output* channel has the mnemonic ?OUCH and in the initial process it is opened to device @TTO. When creating a process, the user can specify alternate input and output devices to which ?INCH and ?OUCH can be opened (packet words ?RCH0 and ?RCH1 in the ?PROC system call).

File Positioning

As soon as you open a channel to a disk file, the system tracks your position in the file with a 32-bit *file pointer.* This pointer is the number of the next byte in the file to be read or written. Normally this pointer is simply incremented for each byte transferred, so that the entire file is processed sequentially. When the pointer is zero, the channel is positioned at the beginning of the file.

Use the ?GPOS system call to determine the file pointer's current value for any channel. You can use the ?SPOS call to change the value of this pointer, thus permitting random access to any byte in the file.

I/O Buffers

As discussed in Chapter 5, MP/AOS provides a software-maintained buffer cache for disk I/O to minimize disk access and to provide efficient sequential I/O. The ?CLOSE system call ensures that all system buffers associated with a particular channel are written (*flushed*) to the disk file. It is good practice to ?CLOSE a file when it is no longer being used.

The buffering mechanism is bypassed when *block-aligned* data is transferred. (See "Dynamic I/O.")

?WRITE/?READ Flush

During the normal ?WRITE to a file, there is no assurance that data is written to the disk, even when ?WRITE returns normally: the data may merely have been transferred to buffers. Typically the data is written to disk either when the disk buffer currently holding the data is needed for buffering other data, or when the file is ?CLOSEd. Since this may not be appropriate for certain applications, MP/AOS provides the FLUSH option on ?WRITE.

When a ?WRITE system call with the FLUSH option returns, the user is assured that all file system data associated with the I/O channel is written to disk. This helps ensure that the state of the file is valid and updated between data transfers.

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

Nonpended I/O

To eliminate the loss of processor time while I/O calls are executing, you can use *nonpended* I/O calls.

Specify a nonpended I/O call by coding the NP option on the call. When you execute the call, instead of suspending your task, the system creates a new task and assigns it the job of executing the I/O call, while the calling task continues its operation. See Figure 9.1.

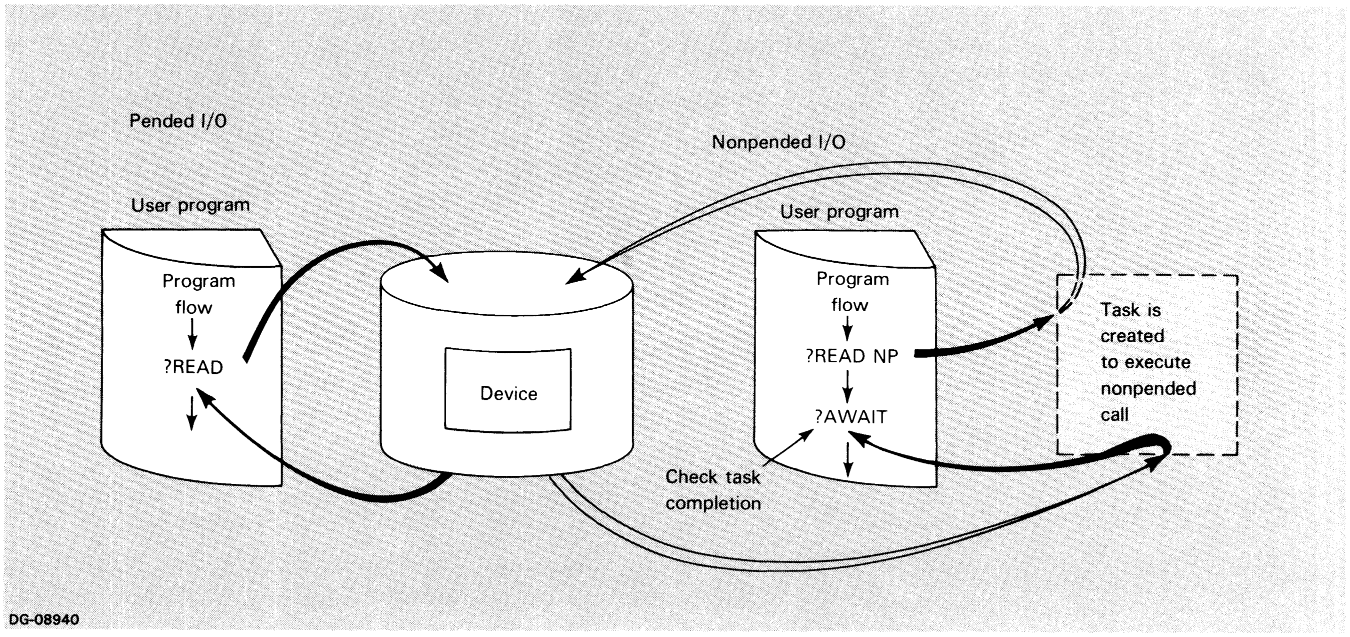


Figure 9.1 Pended and unpended I/O

When you specify the NP option, AC2 returns the task identifier of the system-created task which is executing your I/O. To avoid error when using nonpended calls, the user must specify a sufficient number of tasks (one for each nonpended call) at process creation time (packet word ?RMTC in the ?PROC system call).

You cannot assume that the results of the system call are valid; if, for instance, you read data with a ?READ NP system call, you must still wait for the data to arrive before you can operate on it. You can, however, perform other types of computation while waiting for the new data.

To find out when the nonpended call is complete, execute an ?AWAIT call. This call enables you either to check the nonpended call's progress or to suspend your task until the call is complete. ?AWAIT normally checks the status of a specific nonpended system task; an option allows this call to check the status of any nonpended system tasks.

When issuing ?AWAIT without options, you specify the particular system call to be awaited by supplying a task identifier; this task identifier is the one returned to you in AC2 by the system when you executed the non-pended call.

When the call being executed by the task you specify is ćompleted, ?AWAIT returns the nonpended call's outputs in AC0-2.

The AY option on ?AWAIT allows you to determine when any of your non-pended system calls are completed, rather than only the particular call you specified. In that case, ?AWAIT returns the task identifier of the completed call in AC2. You can then issue another ?AWAIT using the task identifier just returned to obtain the completed call's outputs in the usual fashion.

I/O Techniques

All I/O is performed by means of the ?READ and ?WRITE system calls. Options and packets for these calls allow for a number of different I/O techniques.

MP/AOS transfers data either by byte count *(dynamic I/O)*, or until a delimiter is encountered *(data-sensitive I/O)*.

Normally, data transfers take place between the device and the user's logical address space. However, *extended I/O* (direct segment I/O) permits data to be transferred directly between a memory segment and a disk device. (See Chapter 5 "Memory Management" for more information on memory segmenting).

The amount of data transferred and its placement (i.e., whether or not it is word/block aligned) determine whether or not the data is buffered through the system.

Dynamic I/O

Dynamic I/O permits you to read or write any number of bytes with a single system call. You specify the number of bytes to be transferred in an accumulator; the data is then transferred directly between the file and main memory, subject to buffering by the system.

Block aligned I/O is a special case of dynamic I/O which eliminates system overhead for buffering, resulting in significantly increased speed.

Data on disk devices is divided by hardware into *blocks* of 512 bytes. When you request a data transfer to *part* of a disk block, the system stores the entire block in a system buffer before moving the data to or from this buffer.

If you use the ?READ and ?WRITE calls to transfer entire disk blocks, you eliminate the need for the system to buffer the transfer. To accomplish this, your data must be *block aligned*:

- the file pointer must be a multiple of 512 before the transfer;
- specify a request of 512 or more bytes to read or write;
- specify a buffer that is word aligned in your address space.

The system performs block I/O whenever possible. For example, if the request is for 600 bytes, but the buffer word alignment and file pointer criteria are met, then that portion of the transfer is done by block I/O.

For maximum efficiency, you should not mix dynamic block I/O with conventional dynamic I/O operations.

Data-Sensitive I/O

Data-sensitive I/O is performed by using the ?READ and ?WRITE system calls with the DS option. In this case, the *maximum* number of bytes to be transferred is specified. The system transfers bytes until it encounters a *delimiter*. Default delimiters are defined as bytes containing either a New Line (12_8), Carriage Return (15_8), Form Feed (14_8) or null (0_8). The ?SCHS system call allows you to define any character as a delimiter by specifying a new *delimiter table.* (The maximum number of delimiter tables an MP/AOS system will support is a system generation parameter.) As with dynamic I/O, data is transferred between the file and main memory. After the transfer, the number of bytes moved is placed in an accumulator.

If no delimiter is encountered after the maximum number of bytes has been transferred, the ?READ or ?WRITE call returns error *ERLTL (Line is too long)*.

Extended I/O

Extended I/O is selected by the PKT option to ?READ or ?WRITE. In this case, you specify the address of a packet whose parameters define the transfer.

This form of I/O transfers the data directly between a specified memory segment and a device. Figure 9.2 illustrates.

The memory segment need not be mapped to the user's address space. To determine the segment address for the start of the transfer, the packet specifies a page number, as well as a byte offset.

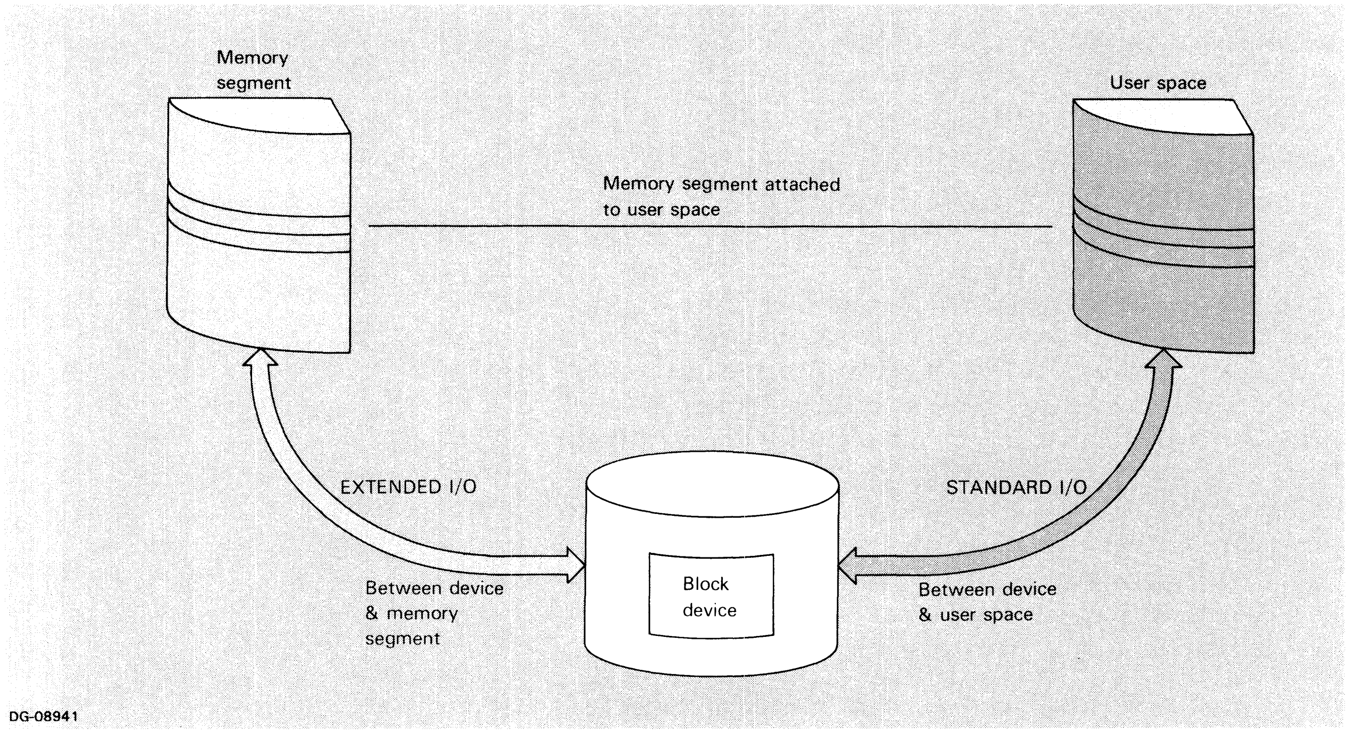

Memory segment
User space
Memory segment attached to user space
EXTENDED I/O
STANDARD I/O
Block device
Between device & memory segment
Between device & user space
DG-08941

Figure 9.2 Standard extended I/O

Extended I/O may be either dynamic or data sensitive. This method is permitted only on disk devices; it is not allowed on character devices. After the end of the transfer, two values are returned: word ?IOBT of the extended I/O packet returns the number of bytes transferred; word ?IOFP is updated to reflect the current position in the file. Thus the user may continue working in the file without modifying the file pointer position.

I/O Device Management

The MP/AOS system divides I/O devices into two categories: those with directory structures *(MOUNTed disks)* and those without directory structures (*character devices*). Character devices include consoles and line printers.

Table 9.1 lists the standard MP/AOS I/O devices with their mnemonics. Note that in a system with several devices of the same type, the mnemonic may be followed by a number, e.g., @DPD0, @LPT1. A complete list of ECLIPSE and microNOVA I/O device codes appears in Appendix D.

	Devices	
Mnem	**ECLIPSE**	**microNOVA**
DPG	20 Mbyte cartridge disk	
DPD	10 Mbyte cartridge disk 315 Kbyte diskette	10 Mbyte cartridge disk
DPH	12.5/25 Mbyte fixed disk 1.25 Mbyte diskette	12.5/25 Mbyte fixed disk 1.25 Mbyte diskette
DPF	50/95/190 Mbyte cartridge disk	
DPX		315 Kbyte diskette
TTI	Console keyboard (input)	Console keyboard (input)
TTO	Console display (output)	Console display (output)
LPT	Line printer	Line printer
MTA	Magnetic tape	Magnetic tape
LP2	Data channel (or PIO) line printer	Data channel (or PIO) line printer
LPB	Data channel line printer	Data channel line printer

Table 9.1 Disk(ette), console, line printer and magnetic tape devices

Disks

The system provides two calls that mount disk devices and dismount them so that you can remove disk units from the system to mount new media on the drives. The ?MOUNT system call introduces a disk to the system. The ?DISMOUNT system call shuts down a disk device in a consistent manner, ensuring that any I/O data still in system memory space will be written out. When the system is started, only the system master device is mounted.

The system performs consistency checks at ?MOUNT and ?DISMOUNT time. A flag on every MP/AOS disk indicates whether it was dismounted properly; i.e., the system did not crash or some other circumstance did not impede dismounting. ?DISMOUNT sets this flag and ?MOUNT tests it. If the flag is not set at ?MOUNT time, you will have to run the disk FIXUP program to restore the disk to a proper state. If this occurs for the system master device when you start the system, the bootstrap loader automatically runs FIXUP. To make sure you have the disk you want, you may also use ?MOUNT to check its label (a user-spccified disk ID name).

NOTE: *You can only ?MOUNT an MP/AOS-formatted disk. If a disk is not properly formatted, use the DINIT utility to prepare it. DINIT is described in* MP/AOS System Generation and Related Utilities *(DGC No. 069-400206).*

If you wish to access a disk without using the MP/AOS file structure, you can ?OPEN it without first ?MOUNTing it. In this case, the disk is treated as a single file with an element size equal to the number of blocks on the disk.

You can use a ?DSTAT call to retrieve status information pertaining to a disk. This call provides such data as the number of blocks in use and the number of I/O errors that have occurred. ?DSTAT may only be used on a ?MOUNTed disk.

Magnetic Tape

MP/AOS supports magnetic tape devices as part of its library. To use a magnetic tape controller, include it in the total count of ?IDEF/?LDEF device control tables (DCT's) specified at system generation. (DCT's are discussed in Chapter 10.)

Magnetic tape devices are supported by the MOVE utility; in most cases you can, therefore, utilize these devices without writing special programs. The MOVE utility is documented in *MP/AOS File Utilities* (DGC No. 069-400204).

For direct access to magnetic tape devices without use of the MOVE utility, a program must be bound with MTA.LB, the tape routine library. This introduces into the program the service routines identifying the magnetic tape controller as a user device. Tape operations are described in Appendix H of this manual.

Character Devices: Terminals

Terminal devices have a number of unique attributes, since they communicate directly with users. Terminals are logically handled as two different devices: the keyboard for input and the printer or CRT for output.

MP/AOS supports data channel line printers via library routines. At system generation, you must specify an ?IDEF/?LDEF DCT for each data channel line printer you wish to include in your system.

You can access data channel line printers via the SPOOLER utility without writing any special code. Should you wish to bypass SPOOLER, bind your program with DCLP.LB, the data channel line printer library that makes the appropriate service routines available.

Data channel line printer support via SPOOLER or via direct user access is documented in *System Generation and Related Utilities* (DGC No. 069-400206).

Console Characteristics

Console *characteristics* are attributes that control the receiving and transmission of data by the console. Table 9.2 summarizes console characteristics.

Set On	Mnemonic	Affects	Meaning when 1
Input/ Output	?CBIN	Both	Binary mode: disables all special control characters; passes all characters exactly as received (8 bits).
Input	?CECH	Output	Echo mode: echoes all typed characters although some receive special handling as described in text.
Input	?CEMM	Output	Echo characters exactly as input: turns off echoing of control characters as ↑x.
Input	?CESC	Input	Escape mode: handles Escape (33_8) the same as CTRL-C CTRL-A.
Input	?CICC	Both	Ignore control characters except delimiters and characters interpreted by the system.
Output	?CLST	Output	List mode: echoes Form-Feeds (014_8)as "^L" to prevent them from erasing CRT screen.
Input/ Output	?CNAS	Both	Non-ANSII-standard console: supports terminals using older standard for control characters by converting Carriage Returns (015_8) into New-Lines (012_8), and vice versa, on input; on output, converts New-Line to Carriage Return, followed by New-Line, followed by null.
Input	?CNED	Output	Do not echo delimiters.
Output	?CST	Output	Simulates tabs: converts all tab characters (011_8) to the appropriate number of spaces; cursor moves to the beginning of the next 8-character tab column.
Output	?CUCO	Output	Convert to uppercase on output.
Output	?C605	Both	6052, 6053 or similar device: uses cursor movement characters to echo Rubout and CTRL-U by erasing characters from the screen. The two characters following a 37_8 on input and a 20_8 on output will be passed through uninterpreted.
Input/ Output	?C8BT	Input	8-bit characters; the default is to mask all input characters to 7 bits, unless in binary mode.

Table 9.2 Console characteristics

Each characteristic is controlled by a bit in the device's *characteristics word.* The system presets console characteristics to the values you specify at system generation; you can later modify these characteristics via system calls. Use ?GCHAR to display the current setting of console characteristics and ?SCHAR to modify any or all of the characteristics words.

Echoing characters is a typical system preprocessing function on a console: normally, all characters received by the system from the keyboard are *echoed* or retransmitted to the display, so that the user can check the input. Most control characters are echoed in the standard way, e.g., ^A for CTRL-A. However, some control characters, such as New Line, are echoed explicitly since they have special meanings to the console. Others are assigned special meanings by the system. See the sections entitled "Control Characters" and "Control Sequences" and their tables.

To ensure compatibility with standard ASCII-7, the system normally sets to 0 the high-order bit of any byte sent to or from a console. Thus character values range from 0 to 177_8. The ?SCHAR system call with the ?C8BT characteristic bit can be set to disable this and transmit/receive eight-bit characters.

The system also echoes Form Feeds as ^L to prevent them from erasing the CRT screen (?CLST characteristic bit), executes Rubouts (?C605 characteristic bit), and converts to uppercase on output (?CUCO characteristic bit). The user can modify any of these characteristics at will.

NOTE: *All special character actions are disabled when you select binary mode for I/O (?CBIN characteristic bit).*

Options to ?SCHAR also allow the user to modify the number of characters per line, the number of lines per page, and as described in the next section, the terminal's hardware characteristics.

Hardware Characteristics

The terminal's hardware characteristics are user-specified when the system is generated. For hardware with programmable characteristics, options to the ?SCHAR system call allow the user to modify these characteristics under software control.

Hardware characteristics consist of the following:

- number of stop bits
- parity type
- code level
- baud rate for Asynchronous/Synchronous Line Multiplexors (ASLM's)
- clock selection for Asynchronous Line Multiplexors (ALM's)

Stop bits are bits used to indicate the end of data transmission. The number of stop bits is user-selectable within the range indicated in Table 9.3.

Parity consists of an optional bit included with each transmitted character for purposes of error checking. Table 9.3 indicates the available types of parity.

Code level specifies the number of data bits per character; Table 9.3 specifies the selectable range.

Number of Stop Bits	Parameter	Parity	Parameter	Code Level	Parameter
1	?C1S	None	?CNPR	5 bits	?C5BC
1.5	?C15S	Odd	?CODD	6 bits	?C6BC
2	?C2S	Even	?CEVN	7 bits	?C7BC
				8 bits	?C8BC

Table 9.3 Programmable hardware characteristics (?SCHAR with HC option)

Baud rate indicates the rate of character transmission. As Table 9.4 indicates, the operating range extends from 50 to 19.2K baud.

Baud Rate for ASLM	Parameter
50	?C0050
75	?C0075
110	?C0110
134.5	?C1345
150	?C0150
300	?C0300
600	?C0600
1200	?C1200
1800	?C1800
2000	?C2000
2400	?C2400
3600	?C3600
4800	?C4800
7200	?C7200
9600	?C9600
19.2K	?C192K

Table 9.4 Baud rate for asynchronous/synchronous line multiplexors (ASLM boards)

Clock selection allows the user to select one of four internal clocks used to establish baud rates of the lines controlled by that clock. See Table 9.5 for a list of clock numbers and parameters.

NOTE: *Baud rate and clock selection are mutually exclusive decisions.*

Clock	Parameter
0	?CCK0
1	?CCK1
2	?CCK2
3	?CCK3

Table 9.5 Clock selection for asynchronous line multiplexors (ALM boards)

Control Characters

The system assigns special functions to certain control characters. These functions are summarized in Table 9.6.

Character	Octal	Function
Null	0	Standard delimiter: signals the end of a data sensitive ?READ or ?WRITE
CTRL-C	3	Starts a control sequence (described below)
CTRL-D	4	Indicates end of terminal input file (not passed to program)
New Line	12	Standard delimiter (like null)
Form Feed	14	Standard delimiter (like null)
Carriage Return	15	Standard delimiter (like null)
CTRL-O	17	Toggles: eliminates output to console; turns console back on
CTRL-P	20	Reserved for future use*
CTRL-Q	21	Restarts output after CTRL-S
CTRL-R	22	Reserved for future use*
CTRL-S	23	Suspends output so you can read material on a CRT screen
CTRL-T	24	Retypes the current line so you can check what you have typed (hardcopy terminals)
CTRL-U	25	Deletes the current input line
CTRL-V	26	Reserved for future use*
Rubout	177	Deletes the last character you typed from the current input line

Table 9.6 Control characters

Reserved characters are ignored except in binary mode.

Control Sequences

A *control sequence* is a CTRL-C followed by one of the characters whose functions are described in Table 9.7.

Character	Octal	Function
CTRL-A	1	Signals a console interrupt, which may be passed to your program (See "Multitasking.")
CTRL-B	2	Causes termination of the program currently running
CTRL-C	3	Reserved for future use*
CTRL-D	4	If the program is being debugged, enters the debugger
CTRL-E	5	Terminates the current program and saves its state in a break file (See "Program Management.")
(Others)	---	No function: character is passed to your program

Table 9.7 Control sequence characters

**Reserved characters are echoed, but not passed to your program except in binary mode.*

Line Printers

The system's handling of line printers is similar to that of console output. Each device has a characteristics word which is a subset of the word for consoles. The system keeps track of the line and page size for line printers just as it does for consoles.

The applicability of various characteristics to line printers is summarized by Table 9.8.

Characteristic	LPT
?CBIN	Used
?CECH	Unused
?CEMM	Unused
?CESC	Unused
?CICC	Used
?CLST	Used
?CNAS	Used
?CNED	Unused
?CST	Used
?CUCO	Used
?C605	Unused
?C8BT	Used

Table 9.8 Line printer device characteristics

System Call Summary

MP/AOS system calls for Input/Output and device management are summarized in Table 9.9.

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	
?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)
?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	
?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)
?SCHS	Set channel specifications	
?SPOS	Set current file position	EF (cause error return on end-of-file on attempt to extend 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 9.9 Input/output and device system calls

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.

Peripheral devices, their input and output capabilities, as well as input and output programming techniques are discussed at length in the following two manuals:

- *User's Manual Programmer's Reference Series, Peripherals* (DGC No. 014-000632-00)
- *Design Engineer's Reference Series, Interface Designer's Manual for NOVA and ECLIPSE Line Computers* (DGC No. 014-000629-01).

Defining a Device Interrupt Service Routine

The ?IDEF system call introduces a user device and its interrupt routine to the system. (The maximum number of user devices MP/AOS will support is specified when the system is generated.) As input to ?IDEF you specify an *interrupt handler definition packet*, which is a block of memory containing the control data summarized in Table 10.1. User-written device service routines reside in user address space.

Word	Mnemonic	Contents
1	?IHND	Address of user device interrupt handler
2	?IMSK	Mask word
3	?ISTK	User interrupt stack address
4	?ISTL	User interrupt stack length
5	?IDAT	Contents of AC2 at interrupt time
6	?IHPR	Reserved

Table 10.1 User device interrupt handler definition packet

The system builds an *internal device control table* (DCT) based on your packet specifications and enters this DCT into its *interrupt vector table*, a hardware defined array. When the system detects an interrupt request, it indexes into the interrupt vector table to locate the correct device control table. The device control table in turn points to the device's interrupt service routine.

The ECLIPSE hardware is capable of implementing up to sixteen levels of priority interrupts. This is done with a 16-bit *priority mask*. Each level of device priority is associated with a bit in this mask. In order to suppress interrupts from any priority level, the corresponding bit in the mask is set to 1. The device's DCT contains the current interrupt service mask (packet word ?IMSK in the ?IDEF system call). Using this value, the Vector on Interrupting Device Code (VCT) instruction updates this mask and therefore makes the implementation of a priority interrupt system a straightforward procedure. (For a discussion of the VCT instruction refer to the Principles of Operation manual appropriate to your processor.)

When a device requests an interrupt, the processor automatically transfers program control to the system's interrupt service routine. This routine retrieves the device code of the interrupting device and saves return information on the stack.

Before transferring program control to the device's service routine, the system also

- loads AC2 with the contents of ?IDAT, a user-defined pointer to a data area (see note below);
- uses the value of packet words ?ISTK and ?ISTL to initialize the stack pointer and frame pointer to the user interrupt stack. This permits the service routine to perform push/pop and similar stack operations;
- takes the current interrupt service mask and inclusively OR's it with the interrupt service mask in the DCT. The OR operation establishes which devices, if any, can interrupt the currently executing interrupt service routine;
- saves the current Load Effective Address (LEF) mode state;
- enables the user map and disables LEF mode and I/O protection, to permit the device service routine to issue input and output instructions.

The user is responsible for restoring the interrupt mask if it has been modified by the device interrupt handling routine.

NOTE: *A comment on the use of ?IDAT is in order. This word is usually used to point to a user-defined data area that describes the custom device or line device. One use of such a data area is to store the device status returned at interrupt time and then to ?IUNPEND a task waiting on device completion (usually, the ID of this task is also found in this user-defined data area).*

Using this technique, interrupt routines are kept short and system interrupt latency is minimized.

The only system calls permitted during a device interrupt service routine are ?IUNPEND, ?STMP, and ?IXIT.

?IUNPEND enables the routine to communicate with other tasks. The system call ?STMP discussed in the following sections sets up data channel map slots to point to a buffer area in user space before a channel transfer is initiated. Interrupts are always enabled after ?IUNPEND and ?STMP.

?IXIT returns control to the system and must be executed to exit from the routine.

MP/AOS restores LEF mode and I/O protection to their former states upon exit from the device service routine.

Use the ?IRMV system call to deactivate device service routines. Your program must deactivate such routines before the system permits it to call another program with the ?EXEC call.

The system automatically deactivates any device interrupt service routines upon program termination. Whenever possible, though, explicit deactivation of such routines by the program via ?IRMV is preferable. If a user device interrupts after its interrupt handling routine has been disassociated from it, the interrupt is handled via the standard system procedure for undefined interrupt processing.

Defining a Line Device Interrupt Service Routine

The ?LDEF system call allows the definition of an interrupt service routine for a single line of an Asynchronous Line Multiplexor (ALM) or an Asynchronous/Synchronous Line Multiplexor (ASLM). With this functionality, the user can elect to control some of the devices connected to the multiplexor, while leaving others under system control.

When a line multiplexor (either ALM or ASLM) is included in the system configuration, MP/AOS builds an internal device control table (DCT) for the multiplexor and enters this DCT into its interrupt vector table. The interrupt service mask for the line multiplexor is included in its device control table.

Any ALM or ASLM line intended for a custom line device must be identified together with its terminal device during the system generation dialogue, and included in the total count of ?LDEF devices requested by SYSGEN, the system generation utility. The system uses this information to allocate space for a user ?LDEF DCT for each user-controlled line.

When your program issues an ?LDEF call, the system builds the actual user ?LDEF DCT containing the line number and the address of its user interrupt service routine, as specified in inputs to the call.

As input to ?LDEF you specify a *line interrupt handler definition packet*, a block of memory containing control data summarized in Table 10.2. The packet format is similar to that used in the ?IDEF call except for the fact that it contains no mask word, because the interrupt service mask is already contained in the DCT for the multiplexor device.

Word	Mnemonic	Contents
1	?LHND	Address of interrupt handler
2	?LSTK	User interrupt stack address
3	?LSTL	User interrupt stack length
4	?IDAT	Contents of AC2 at interrupt time
5	?LHPR	Reserved

Table 10.2 Line interrupt handler definition packet

When a line device interrupts, the system's interrupt service routine locates the ALM or ASLM DCT, and the line number requesting service. Next the system checks whether an ?LDEF has been issued for the device connected to this line (that is, whether the line is user-controlled); if so, the system locates the line's ?LDEF DCT.

Before transferring program control to the device's service routine, the system

- saves return information on the stack.
- loads AC2 with the contents of ?IDAT, a user-defined pointer to a data area (see note below).
- uses the value of packet words ?LSTK and ?LSTL to initialize the stack pointer and frame pointer to the user interrupt stack. This permits the service routine to perform push/pop and similar stack operations.
- takes the current interrupt service mask and inclusively OR's it with the interrupt service mask established by the system in the DCT for the multiplexor device. The OR operation establishes which devices, if any, can interrupt the currently executing interrupt service routine.
- saves the current Load Effective Address (LEF) mode state.
- enables the user map and disables LEF and I/O protection mode to permit the device service routine to issue input and output instructions.

If the interrupt mask has been modified by the ?LDEF interrupt handler, the user is responsible for restoring it.

NOTE: *A comment on the use of ?IDAT is in order. This word is usually used to point to a user-defined data area that describes the custom device or line device. One use of such a data area is to store the device status returned at interrupt time and then to ?IUNPEND a task waiting on device completion (usually, the ID of this task is also found in this user-defined data area).*

Using this technique, interrupt routines are kept short and system interrupt latency is minimized.

The only system calls permitted during a line device interrupt service routine are ?IUNPEND, ?STMP, and ?LXIT. Note that interrupts are always enabled after ?IUNPEND and ?STMP. ?LXIT returns control to the system and must be executed to exit from the line device user service routine.

MP/AOS restores LEF and I/O protection mode to their former states upon exit from the line device service routine.

Use the ?LRMV system call to deactivate line device service routines. Your program must deactivate all such routines before the system permits it to call another program with the ?EXEC call.

The system automatically deactivates any line device interrupt service routines upon program termination. Whenever possible, however, explicit deactivation of such routines by the program via ?LRMV is preferable.

Enabling and Disabling Access to all Devices

The instruction format for LEF (Load Effective Address) and for I/O instructions is identical; hence, LEF or I/O mode must be set to enable the CPU to distinguish between these two classes of instructions. The ?ENBL/?DSBL system calls control the setting of I/O and LEF modes.

No device I/O can occur while the CPU is in LEF mode. To issue I/O instructions anywhere in a program at the task level, a user device driver must, therefore, enable I/O with the ?ENBL command. This permits I/O instructions to be issued to both system and user devices. ?DSBL disables access to all devices. These system calls are valid for the entire program, rather than for the calling task only.

Initially, each process has LEF mode enabled. The user is cautioned that when the CPU is in LEF mode, a user program can use the LEF instruction, but may not issue I/O instructions because they would be interpreted as LEF instructions. Similarly, any LEF instructions issued when LEF mode is disabled are interpreted as I/O instructions.

Under MP/AOS the ?ENBL/?DSBL calls simultaneously control the I/O protection bit and the I/O-LEF mode. When I/O access is enabled, both LEF mode and the I/O protection bit are disabled. Similarly, when I/O access is disabled, LEF mode as well as the I/O protection bit are enabled.

> **WARNING:** *Extreme care must be used when enabling I/O instructions, since doing so allows the user to issue I/O instructions to* any *device.*

Managing Data Channel Map Slots

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

Burst multiplexor I/O requires program control only at the start of each block transfer and is generally faster than data channel I/O. Note that not all user-defined devices have BMC hardware.

Under *data channel I/O,* the system transfers data between the device and memory in blocks at a lower transfer rate than BMC. Data channel I/O requires program control at both the start and end of each block transfer.

Data channel and BMC transfers are performed across *data channel maps* in units whose size is device specific.

Data channel maps are translation tables for the data channel. Another optional map is a translation table for the BMC channel. Devices using data channel maps use a 15-bit logical address. BMC devices use a 21-bit logical address.

All data channel I/O for DGC devices is pre-mapped 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 allows you to access a portion of the data channel or BMC maps and map it to your user address space. Depending on the device's capabilities, allocations spanning several maps are possible.

The following steps summarize the sequence of operations for setting up data channel maps. The remainder of the chapter discusses these operations at greater length.

Step 1: Allocate data channel map slots (?ALMP). (Depending on the device's capabilities, allocations spanning several maps are possible.)

Step 2: Translate user logical address for the start of transfer into a physical page number (?GMRP).

Step 3: Set up data channel map. (Store user physical page number in the appropriate data channel or BMC slot (?STMP), where it serves as a pointer to a buffer in the user address area.)

Step 4: Initiate transfer - enable I/O (?ENBL).

Setting up the data channel maps can be executed from either the driver (*task* level), or the device's interrupt service routine (*interrupt* level); allocating the data channel map slots, obtaining the user physical address, and enabling I/O must be performed at task level.

Data Channel 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.

MP/AOS software convention is to number the slots consecutively starting from 0 for the first slot in map A, through 1151, the last slot in the BMC map. Figure 10.1 summarizes this scheme.

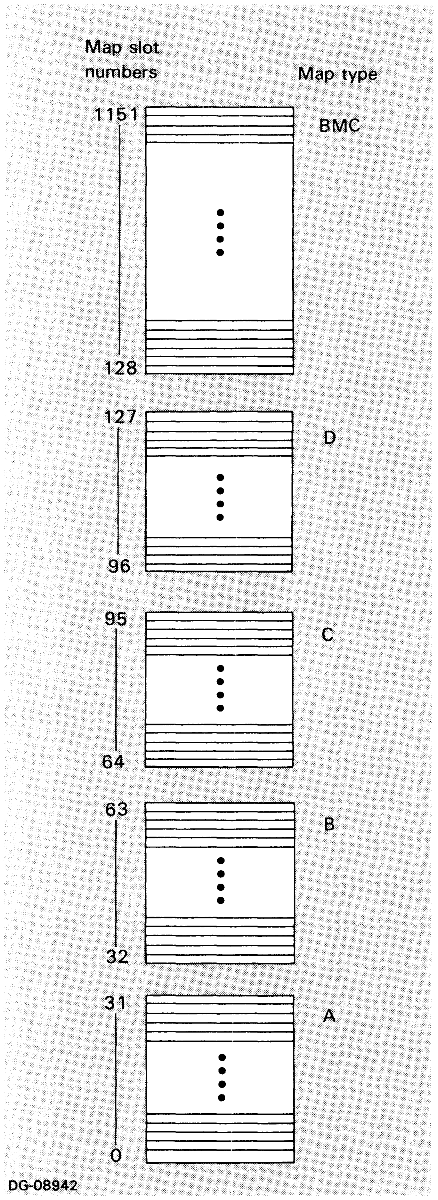


Figure 10.1 Data channel and BMC map slot numbering scheme

Each data channel map slot word corresponds to a 1K word range of logical data channel addresses, from 0 through 1024_{10}. 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 10.2 illustrates.

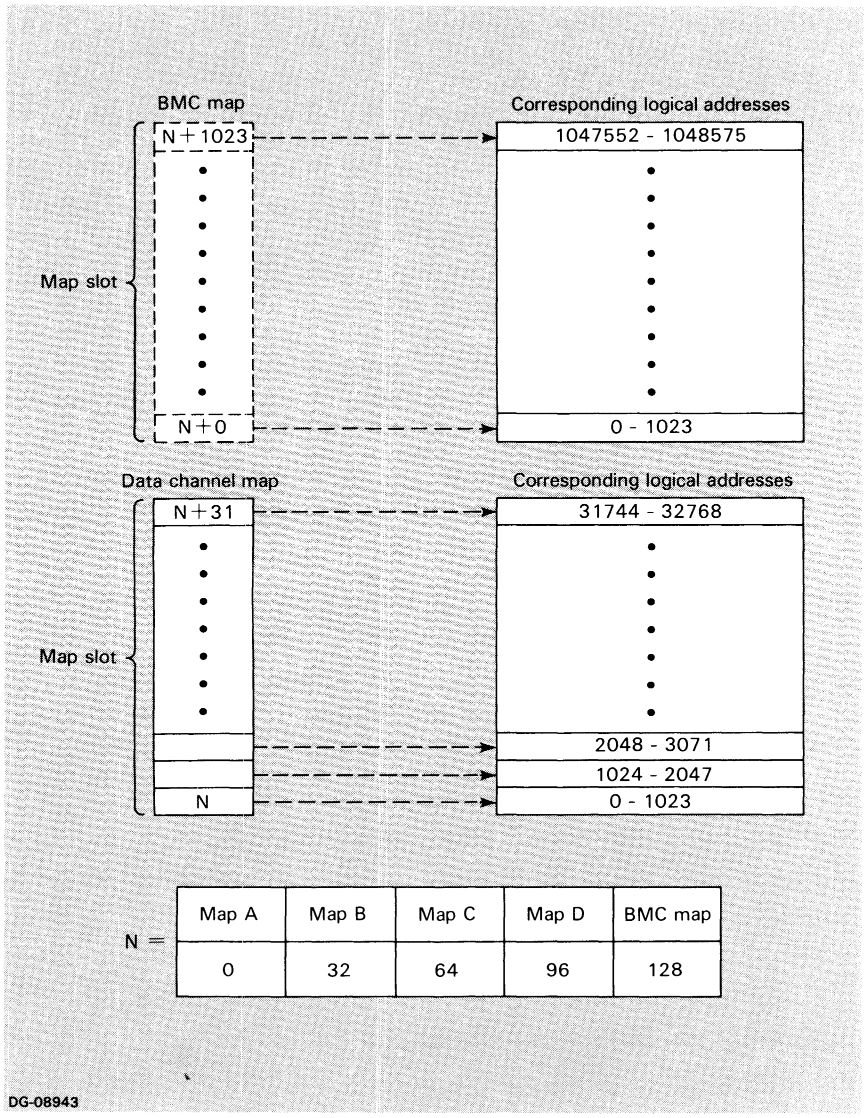


Figure 10.2 Data channel and BMC map slots and their range of corresponding logical addresses

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} - 29696_{10} (70000_8-71777_8).

Data Channel Mapping via System Calls

Once the number of slots required for the particular data transfer is determined, the program issues a request to allocate specific data channel map slots for use by the device. (The number of data channel map slots requested depends on the number of pages to be transferred and on the characteristics of the device.) The ?ALMP system call requests data channel or BMC map slot allocation.

Existing options allow any of the four maps to be specified; using more than one option makes slot allocations spanning several maps possible, provided the device's data channel capabilities allow it.

?ALMP returns the starting data channel map slot number allocated by the system. This number indicates on which map the first slot is allocated, as well as the location of the slot on that map. (Slot 72, for example, is the ninth slot on map C.)

Data channel addresses differ from logical addresses in the user process. The starting map slot number returned is the user's representation of the map slots allocated. The addresses represented by these slots are associated with actual physical pages during a ?STMP call, when the data channel map is actually set up.

Following the allocation of data channel map slots, the user's logical page number from which to transfer data out or in must be translated into a physical page number in memory. (The user address space contains 32 logical pages of 1024 words each.) The ?GMRP system call performs this translation, returning a physical page number. Figure 10.3 illustrates.

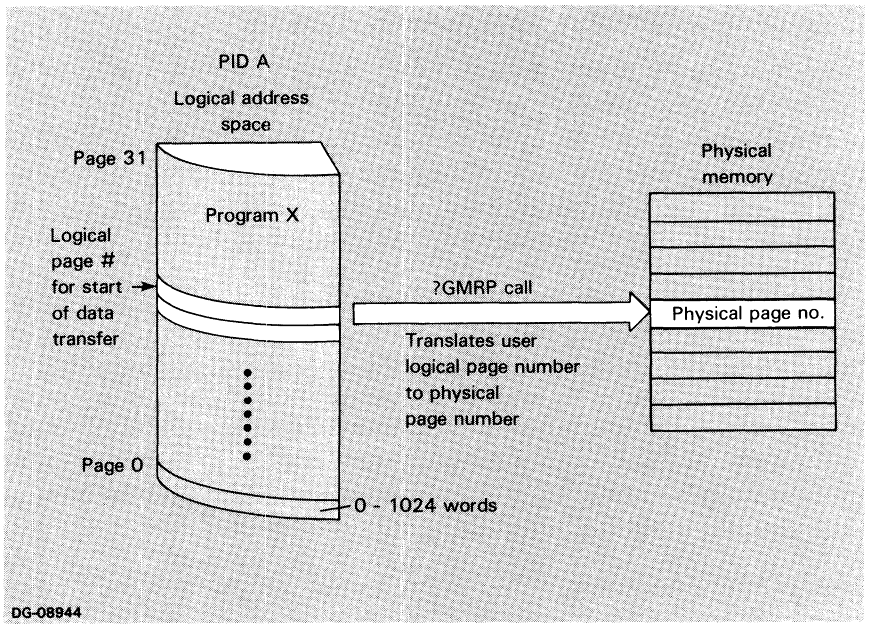


Figure 10.3 ?GMRP call return's physical page number

Upon completion of these steps, data channel mapping can take place. The ?STMP call allows users to request data channel mapping for each map slot previously allocated via ?ALMP.

When ?STMP is issued, the system stores the user's physical page number into the data channel slot number specified. This slot number indicates the range of logical addresses to which the slot in question corresponds.

?STMP fills one slot at a time; it must be reissued for each of the data channel map slots allocated.

Figure 10.4 illustrates the sequence of the three steps just discussed, from data channel map slot allocation through data channel mapping.

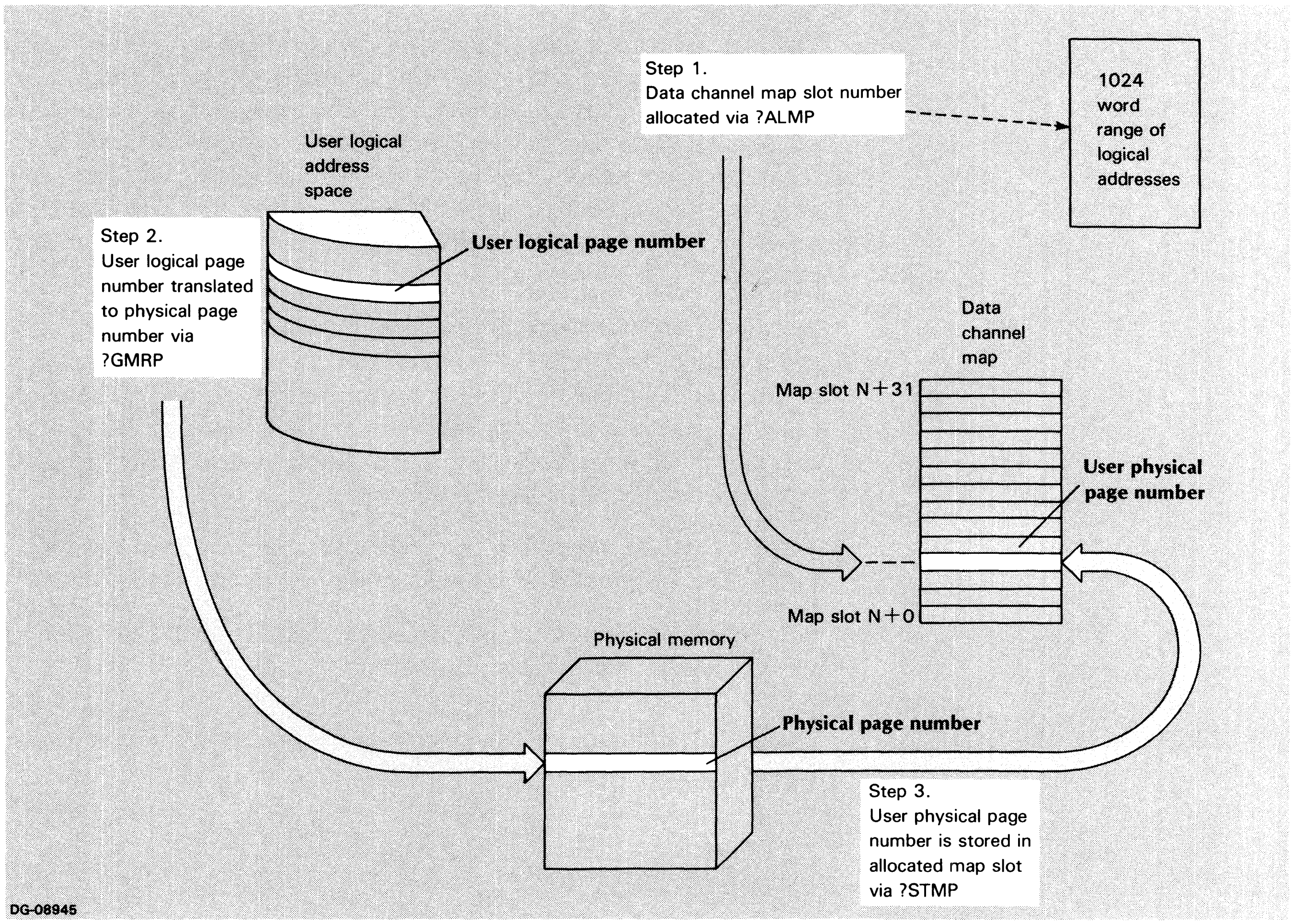

Step 1.
Data channel map slot number allocated via ?ALMP
1024 word range of logical addresses
User logical address space
Step 2.
User logical page number translated to physical page number via ?GMRP
User logical page number
Data channel map
Map slot N+31
User physical page number
Map slot N+0
Physical memory
Physical page number
Step 3.
User physical page number is stored in allocated map slot via ?STMP
DG-08945

Figure 10.4 Sequence of data channel mapping operations

The mapping itself is done by the system. The user must, however, be sure to compute the proper data channel addresses, i.e., to keep track of the slots and their range of data channel logical addresses.

The user program is now ready to issue an I/O instruction. This instruction identifies the device and loads the starting data channel logical address into the accumulator. The starting logical 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.

The starting data channel logical address can be anywhere within the range of logical addresses corresponding to the particular user slot allocated. For transfers which are aligned with page boundaries, the starting data channel logical address is the first address of the range. If, for example, the user has been allocated the fourth map slot, the starting data channel logical address for a page aligned transfer is 6000_8, the first address in the fourth slot range of addresses.

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 must be added to the beginning range address. Using the previous example, if the transfer is to begin at the sixth word of the user's physical page, a starting data channel logical address of $6000_8 + 5_8$ is loaded into the accumulator.

The data is then mapped to the correct user address area referenced by the user physical page number contained in the allocated slot.

Data channel map slots are released with the ?DEMP call, or automatically by program termination.

System Call Summary

Table 10.3 summarizes the system calls available for user device support.

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 10.3 User device support system calls

Miscellaneous System Calls

The calls described in this chapter examine and or change system features, such as the clock and calendar, or perform general functions, such as returning interprogram messages.

Clock/Calendar Calls

The operating system maintains a 24-hour clock and a calendar. A specification in the system generation dialogue allows you to set the clock to any one of several frequencies. See *MP/AOS System Generation and Related Utilities* (DGC No. 069-400206) for the complete system generation dialogue.

The system clock expresses the current time and date in MP/AOS *internal* format, i.e., a 32-bit number representing the number of seconds elapsed since midnight, January 1, 1900. System call ?GTIME returns time and date in internal format; library routines ?CTOD and ?CDAY accept a time and date in 32-bit MP/AOS format and return the hour, minute, and second, and the day, month, and year, respectively.

Alternately, library routines ?GTOD and ?GDAY read, decode, and return system time and date expressed in conventional format, i.e., as hours, minutes, and seconds, and as year, month, and day respectively.

You can set the system time and date to any specified value by using the ?STIME system call and expressing the desired values in MP/AOS internal format, as explained above. (Use library routines ?FTOD and ?FDAY to convert the time and date, respectively, from conventional format to MP/AOS internal format.)

System time and date can also be set by using library routines ?STOD and ?SDAY which accept input in conventional format (i.e., hours, minutes, seconds, and day, month, and year, respectively). Alternately, you can set the date and time by using CLI commands. Table 11.1 summarizes the clock/calendar system calls and library routines and their interrelationship.

Call/Routine	Action	Format of input/output
?CDAY (routine)	Convert date from 32-bit internal	Year, month, day
?CTOD (routine)	Convert time from 32-bit internal	Hours, minutes, seconds
?FDAY (routine)	Convert date to 32-bit internal	32-bit internal format
?FTOD (routine)	Convert time to 32-bit internal	32-bit internal format
?GDAY (routine)	Get system date	Year, month, day
?GTIME (call)	Get system time/date	32-bit internal
?GTOD (routine)	Get system time	Hours, minutes, seconds
?SDAY (routine)	Set system date	Year, month, day
?STIME (call)	Set system time/date	32-bit internal
?STOD (routine)	Set system time	Hours, minutes, seconds

Table 11.1 Clock/calendar calls and routines

Reading a Message

The ?GTMSG call reads into a user-specified buffer any interprogram message transmitted by the most recent ?EXEC, ?PROC, or ?RETURN system call. The system maintains only one message at a time per process. The message can be any string of up to 2047 bytes.

Dictionary of System Calls and Library Routines

This chapter describes the MP/AOS system calls and library routines. *Library routines* are specifically identified as such after their mnemonics and summary descriptions in the dictionary.

Tape commands, which are used in the same way as system calls and library routines, are presented in dictionary format at the end of Chapter 4.

Explanatory Notes

For each entry in this chapter, we give the following information:

- the mnemonic that you place in your program code
- identification of the mnemonic as a *library routine*, if pertinent (Unidentified mnemonics are system calls.)
- a description of the function performed, along with a figure showing the format of the required packet (if any)
- tables specifying inputs, outputs, options, and error returns for each call. The contents of the tables are briefly described below.

Inputs

The inputs table lists information which your program must place in accumulators before executing a given call. Whenever this information is affected by options to the call, the option and its effect are included in the inputs table.

Outputs

The outputs table lists information which will be in the accumulators when control returns to your program. Any accumulators not used for outputs will be unchanged, except for AC3 which is always set to the value of the frame pointer. When outputs are affected by options to the call, the option and its effect are included in the outputs table.

Options

The options table lists and explains options available for each system call.

Errors

The errors table lists the error codes likely to be returned if you use a call improperly. Note that this list is not necessarily exhaustive: under certain conditions, some calls may return codes other than those listed. A complete list of the MP/AOS error codes is contained in Appendix H of this manual.

Error codes are returned in AC0.

For more general information on MP/AOS programming, refer to Chapter 2.

Add a Name to the Searchlist

?ALIST

Appends the specified directory name to your searchlist.

AC0 must contain a byte pointer to the pathname, which must be terminated by a null byte. If AC0 contains 0, the searchlist is cleared; i.e., all entries are removed. The maximum length of a searchlist is five pathnames.

Inputs

AC	Contents
AC0	Byte pointer to pathname of directory (or 0)

Outputs

None

Options

None

Errors

Mnemonic	Meaning
ERDOL	Device is off line
ERFDE	File does not exist
ERFIL	Device read error
ERFTL	Filename too long
ERIFC	Invalid character in filename
ERIFT	Incorrect file type (not a directory)
ERNAD	Non-directory name in pathname
ERPWL	Device write error
ERSTL	Searchlist too long

?ALMP Allocate Data Channel Map Slots

This call directs the system to allocate data channel map slots to the calling program. By using more than one option on a call, the caller can receive a map slot allocation spanning several maps if the data channel capabilities of the device permit it. If the starting map slot number requested conflicts with any of the selected options, ?ALMP returns error ***ERSLT, Slot number error.***

Data channel map options can not be mixed with the BMC (burst multiplexor channel) map option.

In instances where the starting map slot number requested is not available, the system allocates the next available slot number, provided sufficient slots are left in the map to cover the total number of slots requested. If not enough slots are left to satisfy the request, ?ALMP returns error ***ERDCM, Request could not be filled.***

The starting map slot number returned in AC0 indicates on what map and where in that map the first slot is allocated. This map slot number is used as input to the ?STMP call.

See Chapter 10 for discussion of data channel mapping.

Inputs

AC	Contents		
AC0	Number of contiguous map slots requested		
AC1	Lowest acceptable slot number		
	0-	31	Data Channel Map A
	32-	63	Data Channel Map B
	64-	95	Data Channel Map C
	96-	127	Data Channel Map D
	128-	1151	BMC Map

Outputs

AC	Contents
AC0	Starting map slot number. Assignments are identical to the scheme listed in AC1 above.

Options

Mnemonic	Meaning
MA	Data Channel Map A
MB	Data Channel Map B
MC	Data Channel Map C
MD	Data Channel Map D
BMC	BMC Map

Errors

Mnemonic	Meaning
ERDCM	Request could not be filled
ERIDC	Invalid data channel options for microECLIPSE
ERMAP	Not enough map slots
ERSLT	Slot number error

?ASEG Attach a Memory Segment

Attaches the calling program to a segment of memory without mapping it to the caller's address space. Initially a program is both attached and mapped to the segments making up its impure, shared, and overlay areas.

A segment is an area of memory consisting of from 1 to ?MXSP pages (1K word blocks). User created segments are identified and referenced by means of a global segment number assigned when the segment is created. See ?CSEG.

Once a new segment has been created, several programs executing in different processes may attach to it.

The maximum number of attached segments for a given user process is specified at process creation time (see ?RMAS in process definition packet).

Inputs

AC	Contents
AC0	Global segment number

Outputs

None

Options

None

Errors

Mnemonic	Meaning
ERSAA	Segment is already attached
ERSDE	Segment does not exist
ERTMS	Too many segment attaches

Await Completion of a Non-pended System Call

?AWAIT

Used in conjunction with any non-pended system call (NP option) to determine if the call's action is finished.

For example, if you executed a non-pended ?READ, you would use ?AWAIT to determine that the input data was available before you began to operate on it. You specify the particular system call to be AWAITed by a task identifier, which must be the one returned to you in AC2 by the system when you executed the non-pended call.

If the non-pended call is not yet finished, the task that executed ?AWAIT is suspended until the non-pended call completes execution, unless you use the CK option described below.

***NOTE:** You must issue a successful ?AWAIT for every non-pended system call; otherwise a task control block (TCB) will be wasted.*

The AY option causes ?AWAIT to return when *any* non-pended call is completed rather than awaiting the completion of a specific call. In that case, AC2 returns the task ID of the completed task. Your program will then need to issue an ?AWAIT with that task ID so as to receive the completed task's output.

The AY option may be used in conjunction with the CK option.

Inputs

AC	Contents
AC2	Task identifer for non-pended call Option: AY: no input

Outputs

AC	Contents
AC0-2	All accumulators are set to the outputs of the non-pended call. Those not used for outputs are set to their values at the time of the non-pended call.
AC2	Task identifier of completed task if AY option is used.

Options

Mnemonic	Meaning
AY	Return when any non-pended call is completed
CK	Check: if the non-pended call is not yet complete, do not suspend this task; instead, return the ERTIP error code.

Errors

Mnemonic	Meaning
ERTID	Invalid task identifier
ERTIP	Task in progress: the non-pended call is still executing (CK option only)

NOTE: *This call may also return any error codes produced by the non-pended call.*

Block a Process

?BLOCK

?BLOCK blocks the process specified by AC2. That process is not rescheduled until some other process issues an ?UNBLOCK on it.

Blocking a process affects all user tasks in that process. However, system calls already executing in system space run to completion and are blocked upon returning to the blocked program.

?BLOCK has no effect if the specified process is already blocked.

If scheduling for all other processes has been disabled by a ?DRSCH with PRC option, it is reenabled when the executing process issues a ?BLOCK directed at itself, or an ?ERSCH with the PRC option.

Inputs

AC	Contents
AC2	Process ID of process to be blocked (an ID of zero specifies calling process)

Outputs

None

Options

None

Errors

Mnemonic	Meaning
ERPID	Invalid process ID

?BOOT Shut Down the System

Causes the current MP/AOS system to be shut down. All I/O channels are closed; all disk devices are dismounted.

Only the initial process may issue a ?BOOT.

Inputs

None

Outputs

None

Options

None

Errors

Mnemonic	Meaning
ERISC	Invalid system call

Convert System Time/Date to Date *(library routine)* ?CDAY

Accepts a time and a date in 32-bit MP/AOS format and returns the day, month and year. The year is an offset from a base of 1900_{10}.

Inputs

AC	Contents
AC0	High order 16 bits of time
AC1	Low order 16 bits of time

Outputs

AC	Contents
AC0	Day (range $1\text{-}31_{10}$)
AC1	Month (range $1\text{-}12_{10}$)
AC2	Year (minus 1900) result expressed in octal

Options

None

Errors

None

?CLEAR Clear Connection *(IPC Server/Customer call)*

Breaks a single connection (represented by "port number") between the caller and another process, so that no further IPC messages may be transmitted between them.

Any outstanding ?RCV, ?SEND, or ?SD.R calls currently posted by the caller on this port, or by the connected process on its corresponding port are terminated in error ERCBC, *connection broken by customer*, or ERCBS, *connection broken by server*, as appropriate.

?CLEAR can also cause a server's ?RCVA and ?OBITS calls to terminate in error ERSRN, *Server name has been removed and server has no current connections.*

This occurs in cases where the server has performed a ?RMVE and the ?CLEAR is breaking the last remaining connection to this server.

Inputs

AC	Contents
AC0	Port number used to converse with the destination process

Outputs

None

Options

None

Errors

Mnemonic	Meaning
ERPOR	Invalid port

Close an I/O Channel

?CLOSE

Removes the specified I/O channel's connection to a device or file.

If any data from previous ?WRITE calls is in a system buffer, it is written to the file. No more I/O may be performed on the channel until it is reopened.

Inputs

AC	Contents
AC0	Channel number

Outputs

None

Options

Mnemonic	Meaning
DE	Delete the file

Errors

Mnemonic	Meaning
ERICN	Invalid channel number
ERFIL	Device read error
ERPWL	Device write error
ERDOL	Device is off line
ERPRM	Permanent file: cannot be deleted

NOTE: *If you wish data from ?WRITE calls written to the file* before *you close the I/O channel, use ?WRITE with the FL option.*

?CREATE Create a File

Creates an entry for the specified pathname in the directory structure.

The pathname must be terminated by a null byte. You must specify the file type and element size. No attributes are set except for ?ATPM (permanent) and ?ATWR (write-protect) in the case of directories. File attributes and element size are discussed in Chapter 3. Table 3.2 lists file attributes.

Table 12.1 lists the file types available. You may not create new files in the device directory. However, to simplify device-independent programming, the system gives a normal return if a program attempts to ?CREATE a device that already exists.

Mnemonic	Meaning	
?DDIR	Directory	
?DSMN to ?DSMX	Range of values for files used by the system:	
	?DBPG	bootable (stand-alone) program file*
	?DBRK	program break file
	?DIDF	MP/ISAM data file
	?DIXF	MP/ISAM index file
	?DLIB	library file
	?DLNK	link file
	?DLOG	System log file
	?DMBS	MP/BASIC save file
	?DOBF	object file
	?DOLF	overlay file
	?DPRG	program file
	?DPST	permanent symbol table (used by assembler)
	?DSTF	symbol table file
	?DTXT	text file
	?DUDF	general-purpose data file
?DUMN to ?DUMX	Range of values reserved for users	

Table 12.1 File types

**Currently not bootable under MP/AOS.*

NOTES: *If the specified pathname is not fully qualified, the file is created in the working directory. The searchlist is not scanned.*
All directories specified in the pathname must already exist.

Inputs

AC	Contents
AC0	Byte pointer to pathname
AC1	Type of file to create
AC2	File element size in disk blocks; pathname to link, if creating type ?DLNK

Outputs

None

Options

AC	Contents
DE	If the file already exists, delete the old one

Errors

Mnemonic	Meaning
ERDOL	Device is off line
ERFIL	Device read error
ERFTL	Filename too long
ERIFC	Invalid character in filename
ERIFT	Invalid file type
ERNAD	Non-directory name in pathname
ERNAI	File already exists (DE option not used)
ERPRM	Permanent file: cannot be deleted (DE option only)
ERPWL	Device write error
ERSPC	Insufficient file space
ERWAD	Write access denied

?CSEG Create a Memory Segment

Causes a segment of N pages (1K word blocks) of memory to be allocated to the calling program. Segment size may range between 1 and ?MXSP pages. All pages of a newly created segment are zeroed.

?CSEG assigns the newly created segment a global segment number returned in AC0. This global segment number is used to reference the segment in memory management operations such as attaching, detaching, or mapping.

?CSEG causes the newly created segment to be attached to the calling program by means of an implicit ?ASEG call. The new segment is not, however, mapped to a user address space. See ?MSEG for mapping.

User created segments are not swapped in and out by the ?EXEC or ?RETURN calls, nor are they written to break files.

When a new user program is initiated, default memory segments corresponding to its pure, impure, and overlay memory are allocated to it. See Chapter 5 for discussion.

Inputs

AC	Contents
AC0	Number of pages to allocate

Outputs

AC	Contents
AC0	Global segment number

Options

None

Errors

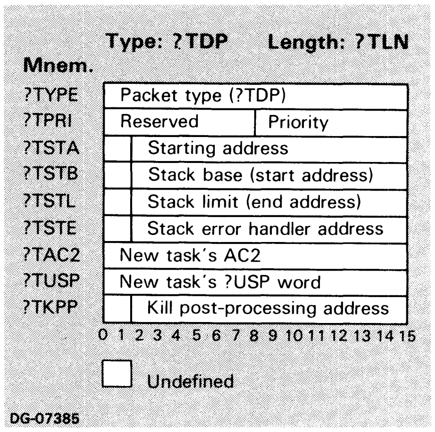
Mnemonic	Meaning
ERNEM	Not enough memory
ERNFS	No free segment
ERTMS	Too many segment attaches

Create a Task

?CTASK

Introduces a new task to the scheduler.

The maximum number of tasks for a given process must be specified by the user when the process is created. See offset ?RMTC in process definition packet. A total of up to 255 tasks for a given process is allowed. AC2 must contain the address of a task definition packet, in which you specify the new task's parameters as defined in Figure 12.1 below.



Mnem.	Type: ?TDP Length: ?TLN
?TYPE	Packet type (?TDP)
?TPRI	Reserved | Priority
?TSTA	Starting address
?TSTB	Stack base (start address)
?TSTL	Stack limit (end address)
?TSTE	Stack error handler address
?TAC2	New task's AC2
?TUSP	New task's ?USP word
?TKPP	Kill post-processing address

Figure 12.1 Task definition packet

If you specify zero in offset ?TSTE, the system will provide a stack error handling routine. In this case, the task will be killed if it overflows its stack.

If you specify zero in offset ?TKPP, the system assumes that you do not wish to perform any kill post-processing for the task. (A kill post-processing routine can perform functions such as deallocating memory used by the task. See discussion in Chapter 8.)

?USP (Unique Storage Position) is one dedicated memory location in lower page zero. Each time a new task is scheduled, the current contents of the ?USP location are saved internally and ?USP is set to the value associated with the new task.

An error is returned if no task control block (TCB) is available to support the new task, *unless* the AW option is specified as described in the Options table.

A default priority of 177_8 (127_{10}) is assigned by the system to a program's initial task.

NOTE: *Task priority (?TPRI) is superseded by process priority. See discussion of scheduling in Chapter 8 ("Multitasking").*

Inputs

AC	Contents
AC0, AC1	Passed to new task
AC2	Address of task definition packet

Outputs

AC	Contents
AC2	Task identifier of the new task

Options

Mnemonic	Meaning
AW	If no TCB is available, wait for one to be freed

Errors

Mnemonic	Meaning
ERNOT	No free TCBs
ERSTS	Invalid stack definition
ERADR	Invalid start address
ERPRP	Invalid priority

Convert System Time/Date to Time of Day *(library routine)* ?CTOD

Accepts a time and date in 32-bit MP/AOS format and returns the hour, minute, and second.

Inputs

AC	Contents
AC0	High order 16 bits of time
AC1	Low order 16 bits of time

Outputs

AC	Contents
AC0	Second (range 0-59_{10})
AC1	Minute (range 0-59_{10})
AC2	Hour (range 0-23_{10} (midnight to 11 pm), (expressed in octal)

Options

None

Errors

None

?DCLR Server Process Declaration *(IPC Server call)*

Declares a name to be associated with the server function provided by the calling process, establishing this process as a server. ?DCLR returns a server number for this name.

A process wishing to communicate with the server process passes the server name to the ?LKUP call.

The server number returned in AC1 serves for use in the ?RCVA, ?PURGE, ?RMVE and ?OBITS calls.

If it wishes to support more than one server function, a process can associate itself with more than one name via several ?DCLR calls. Each server function is then identified by a different server number.

After a successful ?DCLR, the system adds an entry for the calling process in the server table. The size of the server table is a system generation parameter. If the specified server limit is exceeded, ?DCLR returns in error.

Inputs

AC	Contents
AC0	Byte pointer to a name by which the process wishes to be identified. (Name must be terminated by a null byte; its length excluding the null byte can be up to ?svnl bytes. Valid characters include all valid filename characters, except for :, @, =, and ^.

Outputs

AC	Contents
AC1	Server number

Options

None

Errors

Mnemonic	Meaning
ERISN	Invalid server name format
ERSVE	Server already exists
ERSVL	System-wide server limit exceeded

Delay Execution of a Task *(library routine)*

?DELAY

Causes the calling task to be suspended for the length of time specified.

The time, specified in milliseconds, is a 32-bit quantity you place in two accumulators. You may use the ?MSEC library routine to convert hours/minutes/seconds to milliseconds. If you set both accumulators to 0, your task will be delayed for the system default timeout interval (about one minute).

If you set both accumulators to −1, your task will be delayed indefinitely.

This routine uses the ?PEND system call, so if scheduling is disabled, it will be reenabled after suspending the calling task.

Inputs

AC	Contents
AC0	High order 16 bits of the delay time
AC1	Low order 16 bits of the delay time

Outputs

None

Options

None

Errors

None

?DELETE Delete a File

Removes the specified file from the directory structure and returns its disk space to the system.

The pathname must be terminated by a null byte. If the file is open, the filename is removed from the directory, but the disk blocks are not released until all channels open to the file are closed.

If the last or only filename in the pathname is a link, the link itself is deleted, not its resolution.

Directories containing files cannot be deleted, nor can devices be deleted. However, for the sake of compatibility, ?DELETE does not take an error return if you attempt to delete a device.

NOTE: *If the specified pathname is not fully qualified, and the file is not found in the working directory, the ERFDE error return is taken. The searchlist is not* scanned.

Inputs

AC	Contents
AC0	Byte pointer to pathname

Outputs

None

Options

None

Errors

Mnemonic	Meaning
ERFDE	File does not exist
ERPRM	Permanent file: cannot be deleted
ERDID	Directory is not empty
ERNAD	Non-directory name in pathname
ERFTL	Filename too long
ERIFC	Invalid character in filename
ERFIL	Device read error
ERPWL	Device write error
ERDOL	Device is off line

Deallocate Data Channel Map Slots ?DEMP

This call releases data channel map slots held by a program.

Inputs

AC	Contents
AC0	Starting map slot number. Slot assignment is identical to that used during the allocation call (?ALMP): 0- 31 Map A 32- 63 Map B 64- 95 Map C 96- 127 Map D 128- 1151 BMC map
AC1	Number of slots to be deallocated

Outputs

AC	Contents
AC1	Number of slots deallocated

Options

None

Errors

Mnemonic	Meaning
ERSNU	Slot(s) not in use

?DHIS Disable Histogramming *(Histogrammer call)*

Disables histogramming and returns statistics about the terminated histogram, i.e., about the information gathered by the ?EHIS system call. See ?EHIS for a description of the way the histogram is constructed. ?DHIS returns information in a packet with the format described in Figure 12.2 below.

You can disable the histogram while the process being histogrammed is still executing, or you can initiate the histogrammed program by means of a ?PROC with the DB (debug) option, and issue the Await Signal (?WSIG) call setting the signal mask to intercept program termination signals. A ?DHIS system call programmed following the ?WSIG call is then invoked automatically whenever the histogrammed process terminates.

Type: ?HDTP Length: ?HDLN

Mnem.	
?TYPE	Packet type (?HDTP)
?HETH	Elapsed ticks (high)
?HETL	Elapsed ticks (low)
?HSTH	Ticks in system (on user behalf) (high)
?HSTL	Ticks in system (low)
?HUTH	Ticks in user (high)
?HUTL	Ticks in user (low)
?HIDH	Ticks in system idle loop (high)
?HIDL	Ticks in system idle loop (low)

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

DG-08955

Figure 12.2 Histogram disabling packet

Inputs

AC	Contents
AC0	PID of histogrammed process
AC2	Address of packet

Outputs

None

Options

None

Errors

Mnemonic	Meaning
ERNPC	Histogramming not enabled for process specified
ERPID	Invalid PID

?DIR Select a Working Directory

Sets the specified directory to be your current working directory.

The pathname must be terminated by a null byte. If an error occurs, the current working directory is unchanged.

If the specified pathname is not fully qualified, and the directory is not found in the current working directory, the searchlist is scanned.

Inputs

AC	Contents
AC0	Byte pointer to pathname

Outputs

None

Options

None

Errors

Mnemonic	Meaning
ERIFT	Invalid file type (not a directory)
ERFDE	File does not exist
ERNAD	Non-directory name in pathname
ERFTL	Filename too long
ERIFC	Invalid character in pathname
ERFIL	Device read error
ERPWL	Device write error
ERDOL	Device is off line

?DISMOUNT

Remove a Disk From the System

Causes the specified disk device to be disabled from further I/O activity and prepares the disk to be removed from the drive.

The device name must be terminated by a null byte. Any data left in memory from previous I/O is flushed to the disk, and all pointers and directories on the disk are left in an orderly state. A flag is set on the disk to indicate that it was successfully ?DISMOUNTed.

Inputs

AC	Contents
AC0	Byte pointer to device name

Outputs

None

Options

None

Errors

Mnemonic	Meaning
ERFDE	File does not exist
ERNAD	Non-directory name in pathname
ERFTL	Filename too long
ERIFC	Invalid character in filename
ERFIL	Device read error
ERPWL	Device write error
ERDOL	Device off line
ERDAI	Device in use (some I/O channels are open)
ERDNM	Device is not mounted
ERIOD	Specified name is not a device

?DRSCH Disable Task or Process Rescheduling

Disables system scheduling, suspending the execution of all other tasks or processes (PRC option.) If a task issues a ?DRSCH PRC and wishes to suspend all other tasks in the system, it must also issue a ?DRSCH to suspend tasks in its own process.

NOTE: *System calls executing in system space continue execution and are suspended only upon their return.*

Multitasking resumes only when an ?ERSCH call is executed, or when this task executes a ?PEND. If multitasking is already disabled, this call has no effect.

Process rescheduling resumes only when the executing process issues an ?ERSCH PRC or a ?BLOCK directed at itself.

If a process issues a ?DRSCH PRC and all its tasks pend, the system will hang.

You can use ?DRSCH to determine whether multitasking is enabled by using the CK option described below. Since this is a "destructive test," you may then need to execute an ?ERSCH to restore the scheduler's state.

Inputs

None

Outputs

None

Options

Mnemonic	Meaning
CK	Check: if multitasking is already disabled, causes the program to take an error return with code ERSAD
PRC	Disable process scheduler

Errors

Mnemonic	Meaning
ERSAD	Scheduling already disabled (CK option only)

Disable I/O Instructions

?DSBL

This command simultaneously enables the LEF (Load Effective Address) mode and the I/O protection bit in the user's map status word.

When LEF mode is on, user programs may use the Load Effective Address instruction, but may *not* issue I/O instructions. (Any I/O instructions will be interpreted as LEF instructions and can therefore not be carried out.) LEF mode is on when an MP/AOS program is started.

I/O instructions are initially enabled in interrupt handling routines.

Inputs

None

Outputs

None

Options

None

Errors

None

?DSEG Detach From a Memory Segment

Detaches the calling program from a memory segment.* If the segment is currently mapped to the caller's address space it is unmapped, leaving validity protected pages in its place. If the calling program is the only program attached to the segment, the segment is released and its memory is returned to the system. The termination of a program causes an implicit ?DSEG to be issued for every segment attached to that program.

Inputs

AC	Contents
A0	Segment number

Outputs

None

Options

None

Errors

Mnemonic	Meaning
ERSNA	Segment not attached

**A segment is an area of memory consisting of 1 to ?MXSP pages (1K word blocks). User created segments are identified and referenced by means of a global segment number assigned when the segment is created. See ?CSEG.*

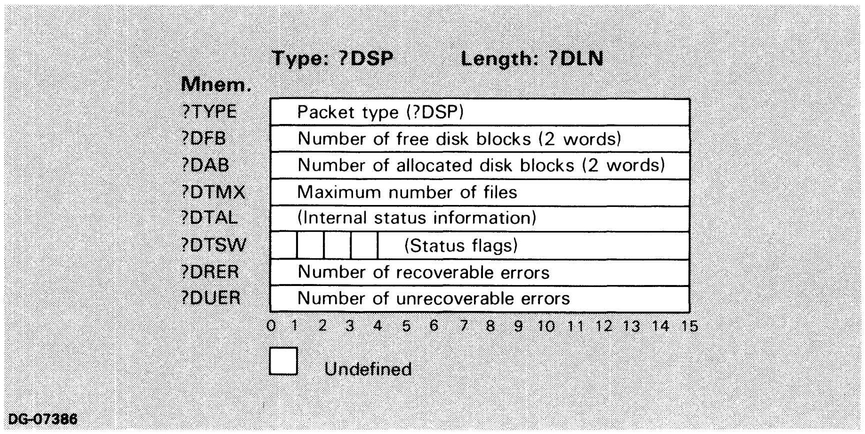
Get a Disk's Status Information

?DSTAT

Retrieves status information about the specified disk.

?DSTAT may be used only on a ?MOUNTed disk.

You specify the disk by its pathname, which must be terminated by a null byte. The status information is placed in a packet, which has the format shown in Figure 12.3 below.



Type: ?DSP Length: ?DLN

Mnem.	
?TYPE	Packet type (?DSP)
?DFB	Number of free disk blocks (2 words)
?DAB	Number of allocated disk blocks (2 words)
?DTMX	Maximum number of files
?DTAL	(Internal status information)
?DTSW	□ | | | | (Status flags)
?DRER	Number of recoverable errors
?DUER	Number of unrecoverable errors

Figure 12.3 Disk status packet

The status flags in the ?DSTW word are described in Table 12.2 below.

Mnemonic	Meaning when 1
?DLE1	Bad primary label block
?DLE2	Bad secondary label block
?DME1	Bad primary MDV (internal information)
?DME2	Bad secondary MDV

Table 12.2 Status flags in ?DSTW word

Inputs

AC	Contents
AC0	Byte pointer to device name of disk
AC2	Address of packet

Outputs

None

Options

None

Errors

Mnemonic	Meaning
ERBTL	Buffer too long
ERDOL	Device is off line
ERFDE	File does not exist
ERFIL	Device read error
ERFTL	Filename is too long
ERIFC	Invalid character in filename
ERIOD	Specified device is not a disk
ERMPR	Invalid packet address
ERPWL	Device write error

Histogram a Process *(Histogrammer call)* ?EHIS

Monitors an executing process and constructs frequency distribution of time spent in address intervals. This call requires a packet of the format illustrated in Figure 12.4 below. The information gathered by ?EHIS can be accessed at any time. A summary of the information is returned by a packet in the ?DHIS call.

Using parameters specified in the packet, ?EHIS places information into a user-specified data buffer and in several named locations that also serve for temporary data storage. Together, the data buffer and the named locations constitute a histogram, or process profile. At regular intervals (also specified in the packet) the system determines where the target process is running and increments a corresponding tally, either at one of the named storage locations or in the data buffer. See Appendix D of Profile documentation (*MP/AOS Debugging and Performance Monitoring Utilities,* DGC No. 069-400205) for an illustration of the data buffer format.

The data buffer consists of one double-word integer for each interval in the range monitored. When a process is being monitored by the system, the value of the program counter (PC) determines which double-word is incremented.

The range of addresses to monitor is specified in packet words ?HSAD and ?HEAD and it can consist of part or all of the user program, up to a maximum length of 77777_8.

The buffer length is specified in packet word ?HLEN. The interval size is determined by the range monitored and the buffer length, according to the formulas:

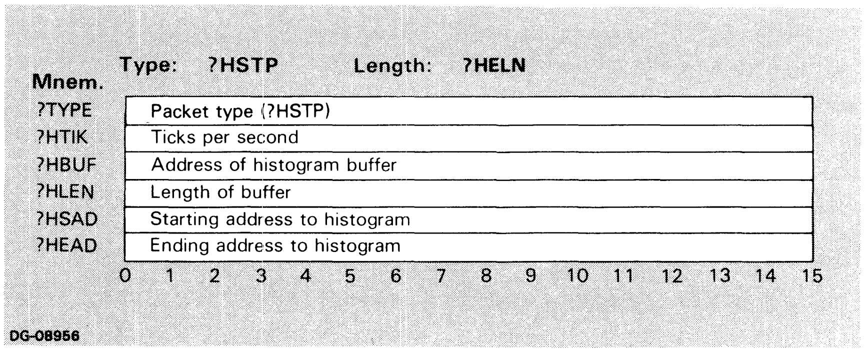
(Buffer length)/2 = Number of intervals
(?HEAD-?HSAD+1)/Number of intervals = Size of intervals

If the interval size determined by the second equation is a fraction, ?EHIS rounds this up to determine the actual interval size.

The data buffer remains in memory, and this is the binding constraint on the length of the buffer. The larger the buffer, the smaller the intervals into which the range is divided. The smaller the interval, the finer the granularity of the observations reported in the buffer. Thus, there is a trade-off between the granularity of the observations and the length of the range monitored.

The number of program interruptions per second is specified by packet word ?HTIK. The system uses the real time clock to generate the process inspections; the actual interval is a multiple of the clock frequency and may not be exactly the value specified in ?HTIK. In any event the actual value used is returned in the ?DHIS packet. (See Appendix B of the Profile program documentation for examples of the calculation of the inspection frequency actually used). The accuracy of the histogram increases with the number of interrupts, but so, too, does the overhead of creating the histogram itself.

The named storage locations keep a tally of total elapsed time, total time spent in the process space (which may be greater than the range), time spent in system space on behalf of the histogrammed process, and time in the system idle loop. This information is displayed in the packet returned by the ?DHIS call.



Type: ?HSTP Length: ?HELN

Mnem.	0 – 15
?TYPE	Packet type (?HSTP)
?HTIK	Ticks per second
?HBUF	Address of histogram buffer
?HLEN	Length of buffer
?HSAD	Starting address to histogram
?HEAD	Ending address to histogram

Figure 12.4 Histogram enabling packet

Inputs

AC	Contents
ACO	PID of process to histogram – cannot be caller's PID
AC2	Address of packet

Outputs

None

Options

None

Errors

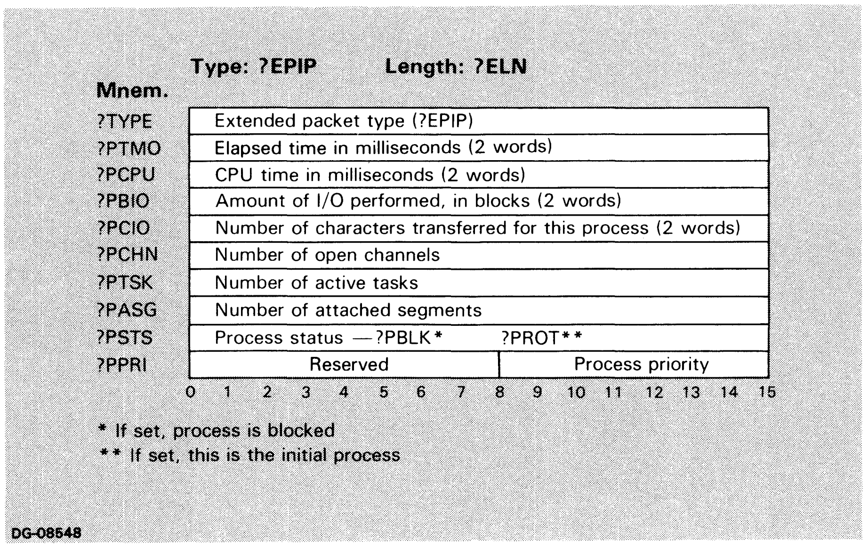
Mnemonic	Meaning
ERPID	Invalid PID
ERMPR	Invalid packet address
ERIPT	Invalid packet type

?EINFO

Get Process Information

Returns a packet containing information about the current state of a process. The packet format is described below in Figure 12.5.

The contents of AC0 specify the ID of the process for which information is requested. Zero in AC0 returns information about the calling process.



Type: ?EPIP   Length: ?ELN

Mnem.		
?TYPE	Extended packet type (?EPIP)	
?PTMO	Elapsed time in milliseconds (2 words)	
?PCPU	CPU time in milliseconds (2 words)	
?PBIO	Amount of I/O performed, in blocks (2 words)	
?PCIO	Number of characters transferred for this process (2 words)	
?PCHN	Number of open channels	
?PTSK	Number of active tasks	
?PASG	Number of attached segments	
?PSTS	Process status —?PBLK*	?PROT**
?PPRI	Reserved	Process priority
	0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15

* If set, process is blocked

** If set, this is the initial process

DG-08548

Figure 12.5 Extended process information packet

Inputs

AC	Contents
AC0	Process ID of process for which information is requested; zero indicates calling process.
AC2	Address of packet

Outputs

None

Options

None

Errors

Mnemonic	Meaning
ERIPT	Invalid packet type
ERMPR	Invalid packet address
ERPID	Invalid process ID

Enable I/O Instructions

?ENBL

Upon completion of this system call, the calling process can issue I/O instructions at the task level.

When I/O is enabled, both LEF (Load Effective Address) mode, and I/O protection are disabled. Any LEF instructions issued while I/O mode is enabled are interpreted as I/O instructions by the hardware.

I/O instructions are always enabled by the system upon entry to a user interrupt handling routine.

Inputs

None

Outputs

None

Options

None

Errors

None

?EQT Set Up System Call

Allows users the option of setting up system calls at runtime. The user specifies the desired system call number and option in AC3 and sets up the contents of accumulators 0 through 2 as defined for the particular call to be executed.

Inputs

AC	Contents
AC0	As defined for selected call
AC1	As defined for selected call
AC2	As defined for selected call
AC3	Number and options of desired system call

Outputs

AC	Contents
AC0	As defined for selected call
AC1	As defined for selected call
AC2	As defined for selected call

Options

None

Errors

As defined for selected call.

Retrieve a System Error Message *(library routine)*

?ERMSG

Reads a message from the MP/AOS error message file :ERMES. If the specified error code has no corresponding message, then the text *Unknown error code n* is returned, where *n* is the error code in octal. If the error file cannot be found, the message *Error code n* is returned.

Inputs

AC	Contents
AC0	Error code
AC1	Byte pointer to message buffer
AC2	Buffer size in bytes

Outputs

AC	Contents
AC2	Actual length of message

Options

None

Errors

Mnemonic	Meaning
ERBTL	Buffer extends into system space
ERDOL	Device off line
ERFIL	Device read error
ERIRB	Buffer too short
ERNMC	No more I/O channels
ERPWL	Device write error

?ERSCH Enable Task or Process Rescheduling

Directs the system scheduler to begin scheduling other tasks or processes (dependent on PRC option). This call has no effect if the requested scheduling mode is already enabled.

Inputs

None

Outputs

None

Options

Mnemonic	Meaning
PRC	Enable process scheduling

Errors

None

Execute a Program

?EXEC

Starts execution of the specified program file. The new program runs at a numerically higher swap level than the initiating program, unless a program chain is specified. A maximum of eight swap levels is possible.

In a program swap the new program runs as a descendant, and the old process state is saved. Memory segment 0 (the impure area) of the calling program is swapped out. Information needed to restore overlay areas and the segment mapping is also recorded in the swap file.

If you specify a program chain, the old process state is not saved.

The following sections apply regardless of whether the executing program is swapped or chained.

The pathname must be terminated by a null byte. Unless the CL option is used, files associated with any open I/O channels are made available to the new program. (However, an error is returned if the calling program has any files opened exclusively and tries to pass its channels.) The CL option closes all channels except ?INCH and ?OUCH. You may pass a message of up to 2,047 bytes to the new program.

The new program is associated with the same process ID as that of the program issuing ?EXEC.

NOTE: *?EXEC returns in error if any user device interrupt handlers are active or if histogramming is enabled.*

Inputs

AC	Contents
AC0	Byte pointer to pathname
AC1	Byte pointer to message (if message length is nonzero)
AC2	Bit 0: 0 = swap 1 = chain Bits 5 - 15: message length (0 if no message)

Outputs

AC	Contents
AC1	Only in case of error return
?ECCP	Error code in AC0 was returned by the called program
?ECEX	Error code was returned by the system; the called program did not run
?ECRT	Error code was caused by ?RETURN which was unable to resume the parent program; in this case, control is passed to the "grandparent" program; i.e., program level is decreased by 2
?ECBK	Error code was returned by the system while trying to write a break file
?ECAB	Error code was returned by the system to indicate an abnormal program termination such as a console abort
AC2	Length of returned message

Options

Mnemonic	Meaning
CL	Close all channels except ?INCH and ?OUCH

Errors

Mnemonic	Meaning
ERABK	Called program terminated by CTRL-C CTRL-E (break file) sequence from console
ERABT	Called program terminated by CTRL-C CTRL-B sequence from console
ERBTL	Message buffer too long
ERDOL	Device off line
EREXS	Attempt to swap beyond program level 8
ERFDE	File does not exist
ERFIL	Device read error
ERFTL	Filename too long
ERIFC	Invalid character in pathname
ERIFT	Invalid file type (not a program file)
ERIRB	Message buffer too short
ERKIL	Reactivate parent process if current process is terminated by ?KILL and was executing at level other than 1
ERNAD	Non-directory name in pathname
ERPCA	Some other task has already issued an ?EXEC or ?RETURN
ERSPC	Insufficient file space
ERPWL	Device write error
ERUIH	User device interrupt handlers are active
ERVNS	Program file is for a different revision of the MP/AOS system

Convert Date *(library routine)*

?FDAY

Accepts input in the form of day, month, and year, and converts it into MP/AOS internal format (a 32-bit number representing the number of seconds elapsed since midnight, January 1, 1900).

Note that the year input in AC2 is an offset from a base of 1900.

Inputs

AC	Contents
AC0	Day (range 1-31_{10})
AC1	Month (range 1-12_{10})
AC2	Year (minus 1900, result expressed in octal)

Outputs

AC	Contents
AC0	High order 16 bits of date
AC1	Low order 16 bits of date

Options

None

Errors

Mnemonic	Meaning
ERANG	Range error

?FSTAT Get a File's Status Information

Returns a packet of information about the specified file.

The file may be specified by channel number, if you have a channel open to it. Otherwise, you can specify the file by its pathname, which must be terminated by a null byte. The status information is placed in a block, which has the format shown in Figure 12.6 below.

Type: ?FSP **Length: ?FLN**

Mnem.	
?TYPE	Packet type (?FSP)
?FTYP	File type
?FATR	Attribute word
?FESZ	File element size (in blocks)
?FTLA	Date and time of last access
?FTLM	Date and time of last modification (2 words)
?FLEN	Length of file (in bytes) (2 words)
	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

DG-07387

Figure 12.6 File status packet

NOTE: *If the specified file is a device, the contents of the ?FESZ, ?FTLA, and ?FTLM words are not applicable. If the file is a character device, the ?FLEN word is also unused.*

If the LNK option is used and the last filename in the pathname is a link, information on the link itself is returned, not on its resolution. In that case, the contents of AC1 are interpreted as a byte pointer to the ?MXLL byte area to receive the link's resolution pathname.

Inputs

AC	Contents
AC0	Byte pointer to pathname Options: CH: channel number
AC1	Options: LNK: Byte pointer to ?MXLL byte area to receive link's resolution pathname
AC2	Address of packet

Outputs

None

Options

Mnemonic	Meaning
CH	AC0 contains a channel number instead of a byte pointer to a pathname
LNK	Do not resolve links

Errors

Mnemonic	Meaning
ERBTL	Buffer too long
ERDOL	Device is off line
ERFDE	File does not exist
ERFIL	Device read error
ERFTL	Filename is too long
ERICN	Invalid channel number
ERIFC	Invalid character in filename
ERMPR	Invalid packet address
ERNAD	Non-directory name in pathname
ERPWL	Device write error

?FTOD Convert Time of Day *(library routine)*

Accepts input in the form of seconds, minutes, hour, and converts it into MP/AOS internal format, i.e., a 32-bit number representing the number of seconds elapsed since midnight, January 1, 1900.

Inputs

AC	Contents
AC0	Seconds (range 0-59_{10})
AC1	Minutes (range 0-59_{10})
AC2	Hour (range 0-23_{10}(midnight to 11pm expressed in octal))

Outputs

AC	Contents
AC0	High order 16 bits of time
AC1	Low order 16 bits of time

Options

None

Errors

Mnemonic	Meaning
ERANG	Range error

Get Device Characteristics

?GCHAR

Places the characteristics word of the specified device (which must be a character device) into an accumulator.

The device name must be terminated by a null byte. See ?SCHAR for a list of characteristics.

Inputs

AC	Contents
AC0	Byte pointer to device name Options: CH: Channel number

Outputs

AC	Contents
AC1	Device characteristics word Options: HC: hardware characteristics LL: number of characters per line PG: number of lines per page RS: characteristics at system boot

Options

Mnemonic	Meaning
CH	AC0 contains a channel number instead of a byte pointer to a pathname.
HC	Return terminal's hardware characteristics in AC1 (for hardware with programmable characteristics only). See ?SCHAR.
LL	Return the number of characters per line in AC1.
PG	Return the number of lines per page in AC1.
RS	Return value of characteristics at the time system was booted.

Errors

Mnemonic	Meaning
ERFDE	File does not exist
ERFTL	Filename too long
ERIFC	Invalid character in filename
ERIFT	Not a character device
ERNAD	Non-directory name in pathname

If the CH option is used, ?GCHAR returns the characteristics for the device open on the channel specified in AC0. An error is returned if the channel is not open on a character device.

If the RS option is used, ?GCHAR returns the device characteristics as they were when the system was booted.

Get the Current Date *(library routine)*

?GDAY

Gets the system time, decodes it into year, month, and day, and returns these values in accumulators. The year is an offset from a base at 1900.

Inputs

None

Outputs

AC	Contents
AC0	Day (range 1-31_{10})
AC1	Month (range 1-12_{10})
AC2	Year (minus 1900; result expressed in octal)

Options

None

Errors

None

?GIDS Get Task Identifiers *(Debugger call)*

Returns the task identifiers of all the tasks in the program being debugged. The list of task ID's provides task numbers for use with the ?RDST and ?WRST calls which allow display and modification of task status for any task in the program being debugged.

Before issuing ?GIDS, the caller must set up a buffer area to receive the task identifiers. The buffer address can be in any area within the caller's logical address space, or within a user-defined segment currently mapped to the calling process, provided that this area is not write protected.

If the buffer length is insufficient to hold all task identifiers, the buffer is filled and no error is returned. (The number of currently defined tasks in the program can be obtained by issuing the ?EINFO system call and reading the value of packet word ?PTSK.)

After ?GIDS has finished executing, AC2 contains the number of task ID's actually returned by the call.

Inputs

AC	Contents
AC1	Address of buffer to receive task identifiers
AC2	Length of buffer in words

Outputs

AC	Contents
AC2	Number of task identifiers returned

Options

None

Errors

Mnemonic	Meaning
ERNDB	No program to debug

Get the Searchlist

?GLIST

Retrieves the contents of your current searchlist into a buffer.

The searchlist is represented by a series of pathnames, separated by commas and terminated by a null byte. All pathnames are fully qualified; i.e., they start at the device directory.

Inputs

AC	Contents
AC0	Byte pointer to buffer
AC1	Length of buffer in bytes

Outputs

AC	Contents
AC1	Length of searchlist (not counting final null byte)

Options

None

Errors

Mnemonic	Meaning
ERIRB	Buffer too short
ERBTL	Buffer too long
ERFIL	Device read error
ERDOL	Device off line

?GMRP Get Physical Page

This call returns a physical page number corresponding to a logical address in the user process.

User input in AC0 is either a global segment number or a local segment number corresponding to user pure, impure, or overlay segments. See Chapter 5 for a discussion of segments.

NOTE: *A logical page is 1024 words long.*

Inputs

AC	Contents
AC0	Local or global segment number
AC2	Logical page number in user process

Outputs

AC	Contents
AC2	Physical page number

Options

None

Errors

Mnemonic	Meaning
ERSNA	Segment not attached
ERMLS	Request longer than segment

Get the Fully-Qualified Pathname

?GNAME

Accepts a filename, pathname, or process ID, and returns a fully-qualified pathname (one that starts at the device directory) corresponding to it. If no such file is found in the current working directory, and no prefixes (@, ^, or =) are present, then ?GNAME resolves the filename through the searchlist looking for the filename. The output pathname is placed in a buffer and terminated with a null byte.

The input filename may contain prefixes; this enables you to find the name of your current working directory by calling ?GNAME with the filename =. You can also use ?GNAME CH to determine the name of the file that is open on a specified I/O channel, ?GNAME PR to determine the name of the currently running program, or ?GNAME PID to determine the program file pathname of another process for which only the process ID is known.

Inputs

AC	Contents
AC0	Byte pointer to filename Options: CH: channel number PID: process ID PR: ignored
AC1	Byte pointer to buffer for pathname
AC2	Length of pathname buffer in bytes

Outputs

AC	Contents
AC2	Length of returned pathname in bytes (not counting final null byte)

Options

Mnemonic	Meaning
CH	AC0 contains a channel number
PID	AC0 contains a process ID
PR	Get the pathname of the calling program (AC0 is ignored)

Errors

Mnemonic	Meaning
ERBTL	Buffer too long
ERDOL	Device off line
ERFDE	File does not exist
ERFIL	Device read error
ERFTL	Filename too long
ERIFC	Invalid character in filename
ERIRB	Buffer too short
ERNAD	Non-directory name in pathname
ERPID	Invalid process ID

Get Next Filename in Working Directory *(library routine)* ?GNFN

Retrieves filenames from the specified directory. Each filename is placed in a buffer in your memory space and terminated by a null byte.

To use this call, you ?OPEN the directory and place the I/O channel number returned by the ?OPEN call into AC0. Then each call to ?GNFN will return one filename. An EREOF (end-of-file) error return is taken after the last filename has been read.

Inputs

AC	Contents
AC0	Channel number
AC1	Byte pointer to filename buffer
AC2	Length of buffer

Outputs

AC	Contents
AC2	Length of returned filename (not counting the terminating null byte)

Options

None

Errors

Mnemonic	Meaning
ERBTL	Buffer too long
ERDOL	Device off line
EREOF	End of file encountered
ERFIL	Device read error
ERIRB	Buffer too short
ERPWL	Device write error

?GPOS Get the File Position

Retrieves the 32-bit file pointer for the specified I/O channel and places it in two accumulators. The file pointer points to the next byte to be read or written.

Inputs

AC	Contents
AC0	Channel number

Outputs

AC	Contents
AC1	High order 16 bits of file pointer
AC2	Low order 16 bits of file pointer

Options

None

Errors

Mnemonic	Meaning
ERDOL	Device off line
ERFIL	Device read error
ERICN	Invalid channel number
ERIOD	Invalid operation for device
ERPWL	Device write error

Get File Attributes

?GTATR

Places the attribute word and type number of the specified file in two accumulators.

The pathname must be terminated by a null byte. Tables 12.3 and 12.4 list file attributes and file types respectively.

If the CH option is used, attributes are returned for the file open on the channel specified in AC0.

If the LN option is used, AC1 and AC2 return the length of the file in bytes instead of the file attribute and the file type.

Inputs

AC	Contents
AC0	Byte pointer to pathname or channel number if CH option is used

Outputs

AC	Contents
AC1	Attribute word
AC2	File type
AC1-2	Options LN: File byte length (AC1-high order 16-bits, AC2-low order 16 bits)

Options

Mnemonic	Meaning
CH	Input a channel number instead of a byte pointer to a pathname in AC0
LN	Return file byte length instead of file attribute and file type in AC1-2

Errors

Mnemonic	Meaning
ERDOL	Device off line
ERFDE	File does not exist
ERFIL	Device read error
ERFTL	Filename too long
ERIFC	Invalid character in filename
ERNAD	Non-directory name in pathname
ERPWL	Device write error

Mnemonic	Meaning
?ATPM	Permanent: the file may not be deleted or renamed while this bit is set to 1; set by the system for directories and root directories of disks
?ATRD	Read protect: this file may not be read
?ATWR	Write protect: this file may not be written; set by the system for directories and root directories of disks.
?ATAT	Attribute protect: the attributes of this file may not be changed. (This bit is used by the system for devices and root directories of disks only.)

Table 12.3 File attributes

Mnemonic	Meaning	
?DDIR	Directory	
?DSMN to ?DSMX	Range of values for files used by the system:	
	?DBPG	Bootable (stand-alone) program file*
	?DBRK	program break file
	?DIDF	MP/ISAM data file
	?DIXF	MP/ISAM index file
	?DLIB	library file
	?DLNK	link file
	?DLOG	System log file
	?DMBS	MP/BASIC save file
	?DOBF	objective file
	?DOLF	overlay file
	?DPRG	program file
	?DPST	permanent symbol table (used by assembler)
	?DSTF	symbol table file
	?DTXT	text file
	?DUDF	general-purpose data file
?DUMN to ?DUMX	Range of values reserved for users	

Table 12.4 File types

**Currently not bootable under MP/AOS*

?GTIME Get the Current System Time and Date

Gets the current time and date, in MP/AOS internal format.

Internal format is a 32-bit number representing the number of seconds elapsed since midnight, January 1, 1900. You may also use the ?GDAY and ?GTOD library calls to retrieve this number in decoded form.

Inputs

None

Outputs

AC	Contents
AC0	High order 16 bits of system time
AC1	Low order 16 bits of system time

Options

None

Errors

None

Get an Interprogram Message

?GTMSG

Reads the current interprogram message into a buffer.

This message may have been transmitted by an ?EXEC, ?RETURN or a ?PROC, whichever occurred most recently. The system maintains only one message at a time per process.

The message may be any string of up to 2,047 bytes. If you a specify a buffer that is too short, your program will take the error return but AC1 will contain the actual message length; thus, you can try again after allocating more memory.

Inputs

AC	Contents
AC0	Byte pointer to message buffer
AC1	Length of buffer in bytes

Outputs

AC	Contents
AC1	Actual length of message

Options

None

Errors

Mnemonic	Meaning
ERIRB	Buffer too short
ERBTL	Buffer too long

?GTOD Get the Current Time of Day *(library routine)*

Gets the system time, decodes it into hours, minutes, and seconds, and returns these values in accumulators. The hour ranges from 0 to 23.

Inputs

None

Outputs

AC	Contents
AC0	Seconds (range 0-59_{10})
AC1	Minutes (range 0-59_{10})
AC2	Hours (range 0-23_{10}) (midnight to 11 pm) (expressed in octal)

Options

None

Errors

None

Get Process Identity Numbers of all Processes in the System

?GTPID

Returns the process ID's of all the processes in the system into a user-specified buffer.

To determine the buffer size for all process ID's, use system call ?SINFO which returns the number of concurrent processes allowed on the system (packet word ?SPRC). One word of buffer space is needed for each process ID.

If the buffer is smaller than the number of processes in the system, it will be filled with the first *n* PID's where *n* is the value input in AC2.

?GTPID effectively returns a snapshot of the system and there is no guarantee that any specific process will continue to exist after the call returns.

Inputs

AC	Contents
AC1	Address of buffer
AC2	Length of buffer in words (use ?SINFO to determine size of buffer)

Outputs

AC	Contents
AC2	Actual number of process identifiers written in the buffer

Options

None

Errors

Mnemonic	Meaning
ERIPT	Invalid packet type
ERMPR	Invalid packet address

?IDEF Define an Interrupt Handling Routine

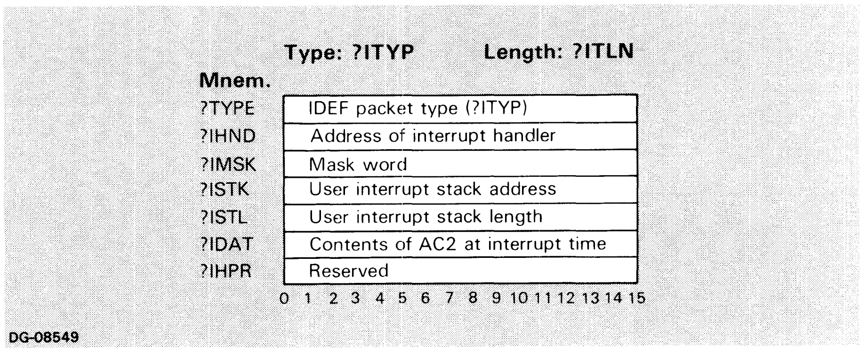
Informs the system that your program will handle interrupts from the specified device. You must specify an interrupt handler packet for the device. The format of this packet is shown in Figure 12.7.

User device handlers execute in user process space.

Packet word ?IMSK contains the interrupt mask word to be OR'd with the current interrupt mask so as to establish the new level of priority interrupts. The user is responsible for maintaining the interrupt mask while in the device interrupt handling routine.

On entry to the service routine, the system enables the user map, and disables LEF (Load Effective Address) mode to allow the device handler to issue I/O instructions. ?IDAT, whose value can be any user-defined data, is placed in AC2, and the stack pointer and frame pointer to the user interrupt stack are initialized, using packet words ?ISTK and ?ISTL.

NOTE: *One user interrupt stack is required for each ?IDEF call.*



Type: ?ITYP **Length: ?ITLN**

Mnem.	
?TYPE	IDEF packet type (?ITYP)
?IHND	Address of interrupt handler
?IMSK	Mask word
?ISTK	User interrupt stack address
?ISTL	User interrupt stack length
?IDAT	Contents of AC2 at interrupt time
?IHPR	Reserved

Figure 12.7 Interrupt handler definition packet

Inputs

AC	Contents
AC0	Device code
AC2	Address of packet

Outputs

None

Options

None

Errors

Mnemonic	Meaning
ERADR	Invalid routine address
ERDVC	Invalid device code
ERIPT	Invalid packet type
ERMPR	Invalid packet address
ERUIH	Service routine already defined for this device

?IFPU Initialize for Floating Point

Indicates that the calling program is going to use floating point. This call must be issued prior to any floating point instruction. ?IFPU clears the floating point status register without affecting the floating accumulators, and it causes the floating point status to become part of the task state for all tasks in the calling program. The initial contents of the floating point accumulators are undefined.

> **WARNING:** *Failure to issue this call prior to the use of floating point instructions will yield indeterminate results.*

Inputs

None

Outputs

None

Options

None

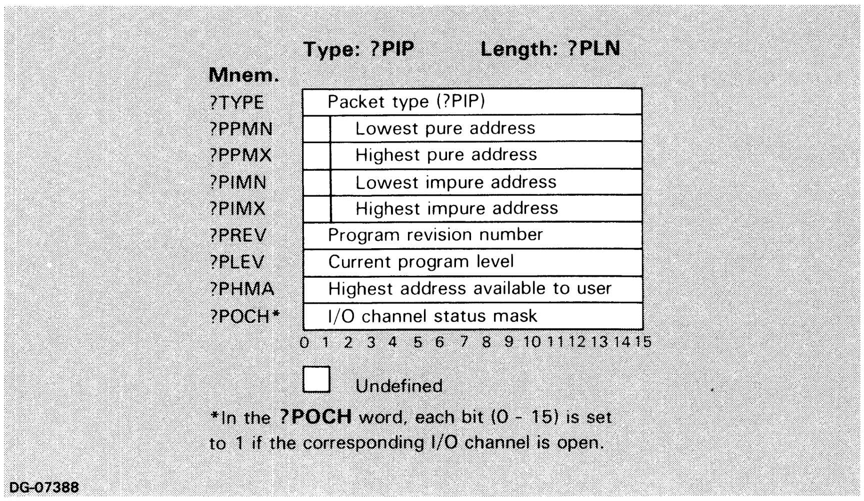
Errors

None

Get Program Information

?INFO

Retrieves a packet containing information about a program's current memory allocation and running state. The format of the packet is given in Figure 12.8 below.



Type: ?PIP Length: ?PLN

Mnem.	
?TYPE	Packet type (?PIP)
?PPMN	□ Lowest pure address
?PPMX	□ Highest pure address
?PIMN	□ Lowest impure address
?PIMX	□ Highest impure address
?PREV	Program revision number
?PLEV	Current program level
?PHMA	Highest address available to user
?POCH*	I/O channel status mask

□ Undefined

*In the **?POCH** word, each bit (0 - 15) is set to 1 if the corresponding I/O channel is open.

DG-07388

Figure 12.8 Program information packet

Pure memory, whose lower and upper bounds are specified by the contents of packet words ?PPMN and ?PPMX, is comprised of two areas: *overlay* memory and *shared* memory. Thus, ?PPMN (lowest pure address) refers to the lowest overlay address or to the beginning of the shared area if no overlay is used. ?PPMX (highest pure memory address) refers to the highest address of shared memory or to the end of the overlay area if there is no shared area. If the program uses neither shared nor overlay memory areas, the contents of ?PPMN and ?PPMX are undefined. See Figure 5.3.

In the ?POCH word, each bit (0-15) is set to 1 if the corresponding I/O channel is open. Information is provided for the first 16 channels only.

Inputs

AC	Contents
AC0	Option: PID: ID of process for which information is requested
AC2	Address of packet

Outputs

None

Options

Mnemonic	Meaning
PID	Retrieve information for process whose ID is specified in AC0

Errors

Mnemonic	Meaning
ERIPT	Invalid packet type
ERMPR	Invalid packet address
ERPID	Invalid process ID

Pend Awaiting Interrupt Activity

?IPEND

This call allows the pending of a task with interrupts off. If interrupts have been disabled by the user process, ?IPEND guarantees that the pend is internally queued before interrupts are re-enabled. In all other respects, ?IPEND functions in the same manner as ?PEND. Interrupts are reenabled after the completion of ?IPEND.

The calling task is suspended from execution until a specified event occurs. The event is defined by a 16-bit number which can be used by another task in an ?UNPEND, or ?IUNPEND call. Event numbers must be greater than or equal to 0 and less than or equal to ?EVMAX. When execution resumes, the system passes a message word from the task which executed the ?UNPEND or ?IUNPEND.

The calling task can also resume execution in response to an ?UNPEND ID or ?IUNPEND ID call, or after a timeout interval elapses. The length of the interval in milliseconds is specified as a 32-bit number in two accumulators. You can also request the system default timeout interval (about one minute) by setting both accumulators to zero.

NOTE: *An ?IPEND call with a timeout value of -1 will pend indefinitely.*

Inputs

AC	Contents
AC0	Event number
AC1	High order word of timeout duration
AC2	Low order word of timeout duration

Outputs

AC	Contents
AC0	Message word from ?UNPEND/?IUNPEND

Options

None

Errors

Mnemonic	Meaning
ERTMO	Timeout interval has elapsed
EREVT	Invalid event number

Remove an Interrupt Handling Routine

?IRMV

Informs the system that your program will no longer handle interrupts from the specified device. Any further interrupts from the specified device are treated as undefined interrupts.

Inputs

AC	Contents
AC0	Device code

Outputs

None

Options

None

Errors

Mnemonic	Meaning
ERDVC	Invalid device code
ERNUI	No handling routine currently defined, or you attempted to remove the system's control of a standard I/O device

?IUNPEND Unpend a Task from Interrupt Handling Routine

Resumes execution of the specified task. This call functions in a manner identical to the ?UNPEND call. The difference is that only ?IUNPEND may be used by an interrupt handler. ?IUNPEND may not be used at any other time.

You can specify the task to be unpended either by its identifier or by a 16-bit event number. Event numbers must be greater than or equal to ?EVMIN and less than or equal to ?EVMAX.

If you specify an event number that several tasks are waiting for, only one task is unpended, unless you use the BD option to unpend all waiting tasks.

The system unpends tasks on event on a first in, first out basis: the first task pended is the first unpended, regardless of time remaining in its timeout interval.

You can also specify that the unpended task take the error return from its ?PEND or ?IPEND call.

Inputs

AC	Contents
AC0	Message word to unpended task
AC2	Event number or task identifier if ID option is used

Outputs

AC	Contents
AC0	Number of tasks unpended

Options

Mnemonic	Meaning
BD	Unpend all tasks waiting for this event
ER	Causes the unpended task(s) to take the error return from the ?PEND or ?IPEND calls
ID	AC2 contains a task identifier, not an event number

NOTE: *Do not specify the BD and ID options together.*

Errors

Mnemonic	Meaning
EREVT	Invalid event number
ERTID	Invalid task identifier

?IXIT Exit From an Interrupt Handling Routine

Returns the system to normal operation after completion of a user interrupt handler. All interrupt service routines *must* exit by this call.

Inputs

None

Outputs

None

Options

None

Errors

None

Terminate a Program

?KILL

Terminates the program whose process ID is specified in AC2, and returns all resources used by that program.

If the terminating program is executing

- at *swap level 1* within the *initial process*: the system is shut down.
- at *swap level 1* within any except the initial process: both process and program terminate.
- at any swap level higher than 1: the *parent* program is reactivated with the error code ERKIL. See ?EXEC.

If the BK option is used, the program and its state information are written to a *break file* in the system working directory (:?SYSDIR). The name of the break file is composed of a question mark (?) followed by seven digits and a .BRK extension, thus:

?xxxxxxx.BRK

The first six digits following the ? represent the PID (process identifier) of the process within which the terminating program is running. The seventh digit represents the program execution level (swap level) of the terminating program.

NOTE: *See ?RETURN for another method of terminating the calling program.*

Inputs

AC	Contents
AC2	ID of process whose current program is to be killed; if 0, kill calling program

Outputs

None

Options

Mnemonic	Meaning
BK	Create a break file of the current program executing within the specified process

Errors

Mnemonic	Meaning
ERDOL	Device off line
ERFIL	Device read error
ERPID	Invalid process ID
ERPWL	Device write error

Kill a Task

?KTASK

Terminates execution of the specified task.

If a kill post-processing routine* is defined for the task, it will be executed. If the killed task has an outstanding system call, that call will be aborted; the degree of completion that the outstanding call reaches is undefined.

Inputs

AC	Contents
AC2	Task identifier, or zero to kill this task

Outputs

None

Options

None

Errors

Mnemonic	Meaning
ERTID	Invalid task identifier

**A kill post-processing routine can perform functions such as deallocating memory used by the task. See discussion in Chapter 8.*

?LDEF Define a Line Interrupt Handling Routine

Informs the system that your program will handle interrupts from the specified line device. The line dedicated to the user device must have been specified at system generation time.

?LDEF is intended for use with Asynchronous and Asynchronous/Synchronous Line Multiplexors (ALM's and ASLM's). This call allows some of the devices connected to the multiplexor to be controlled by the user, while others remain under system control.

Custom line device service routines execute within the user process space.

You must specify a line interrupt handler packet in the format shown below for the line device.

Before transferring control to the line device's user service routine, the system enables the user map and disables LEF (Load Effective Address) and I/O protection mode to allow the device handler to issue I/O instructions. Packet word ?IDAT, whose value can be any user-defined data, is placed in AC2; the stack pointer and frame pointer to the user interrupt stack are initiated, using packet words ?LSTK and ?LSTL.

The user line interrupt routine is responsible for clearing the interrupt.

The only system calls permitted at interrupt time are ?IUNPEND, ?STMP and ?LXIT.

Type: ?LTYP Length: ?LTLN

Mnem.	
?TYPE	LDEF packet type (?LTYP)
?LHND	Address of interrupt handler
?LSTK	User interrupt stack address
?LSTL	User interrupt stack length
?IDAT	Contents of AC2 at interrupt time
?LHPR	Reserved

DG-09101

Figure 12.9 Line interrupt handler definition packet

Inputs

AC	Contents
AC0	Device code Bit 0: 0 = Receiver interrupt 1 = Transmitter interrupt
AC1	Line number
AC2	Address of packet

Outputs

None

Options

None

Errors

Mnemonic	Meaning
ERADR	Invalid routine address
ERDVC	Invalid device code
ERDAI	Device is in use
ERIDF	Out of ?IDEF DCT's
ERIPT	Invalid packet type
ERMPR	Invalid packet address
ERUIH	Service routine already defined

?LKUP Look Up a Server Process *(IPC Customer call)*

Establishes the caller as the customer of the server process looked up. Returns a port number to be used in communicating with this server process.

This call allows the calling process to establish a connection with a server process. The caller specifies a server process name and is returned a port number. This port number is the mechanism whereby the caller addresses subsequent communication to this server process.

The server name input into AC0 is that used by the server process in the ?DCLR call.

If the server process being looked up no longer exists (program termination, ?EXEC, ?RMVE calls issued by the server), ?LKUP returns in error.

Successful completion of ?LKUP establishes a connection between the two parties causing the system to add an entry to its connection table. The maximum number of permissible connections is a system generation parameter. If this limit is exceeded, ?LKUP returns in error.

Inputs

AC	Contents
AC0	Byte pointer to a server name to look up. (Name must be terminated by a null byte; its length excluding the null byte can be up to ?svnl bytes. Valid characters include all valid filename charcters, except for :, @, =, or ^).

Outputs

AC	Contents
AC0	Port number to be used for communicating with the server process

Options

None

Errors

Mnemonic	Meaning
ERSRV	Invalid server
ERISN	Invalid server name format
ERCXL	System wide connection limit exceeded

Remove a Line Interrupt Handling Routine

?LRMV

Informs the system that your program will no longer handle interrupts from the specified line device. Any further interrupts from the specified line device are serviced by the system.

Inputs

AC	Contents
AC0	Device code
AC1	Line number

Outputs

None

Options

None

Errors

Mnemonic	Meaning
ERDVC	Invalid device code
ERNUI	No handler currently defined

?LXIT Exit from a Line Interrupt Handling Routine

Returns the system to normal operation after completion of a user line interrupt service routine. All service routines for user line devices must exit by means of this call.

Inputs

None

Outputs

None

Options

None

Errors

None

Change Impure Memory Allocation

?MEMI

Allocates or releases sections of the program impure memory area (segment 0). See Chapter 5 for discussion of segments.

Memory is always added or removed at the top of the impure area. Use ?INFO to determine the maximum additional impure area available: (area between ?PIMX (current highest impure address) and ?PPMN (lowest pure address, −1. Pure memory consists of the shared and overlay areas).

NOTE: *?MEMI specifies memory operations in word units, whereas the hardware allows memory operations within an address space only in page (1K word) multiples.*

If the user's current impure is exactly N pages, a request to ?MEMI of a single additional word results in the additional allocation of an entire page. The definition of user impure (segment 0) now reflects the actual page added; ?MEMI and ?INFO, on the other hand, reflect only the addition of the single word requested. The user is advised to access only the actual memory requested with ?MEMI.

Inputs

AC	Contents
AC0	Number of words to allocate (if positive) or release (if negative)

Outputs

AC	Contents
AC1	New highest impure address

Options

None

Errors

Mnemonic	Meaning
ERMEM	Invalid request: attempt to acquire or release too much memory

?MOUNT Introduce a Disk to the System

Prepares the specified disk device for I/O.

This call must be executed before any directories on the disk can be accessed.

The system checks a flag on the disk to see if it was properly ?DISMOUNTed. If it was not, your program takes the error return with code ERFIX, and you must run the Disk FIXUP program.

The system can also check the disk label (or disk ID), if any, to verify that the correct disk is mounted. (Disk ID is optionally assigned by the user during disk initialization. See Chapter on DINIT in *MP/AOS System Generation and Related Utilities,* DGC No. 069-400206.)

A nonzero value in AC2 causes the system to return the disk ID, regardless of whether or not an ID check was requested. The value of AC2 is read as the byte address of a buffer into which the system returns the disk ID (terminated by a null byte).

Inputs

AC	Contents
AC0	Byte pointer to device name
AC1	Byte pointer to disk ID, or zero to suppress ID check
AC2	If nonzero, byte pointer to buffer to receive the disk ID

Outputs

None

Options

None

Errors

Mnemonic	Meaning
ERDOL	Device off line
ERFDE	File does not exist
ERFIL	Device read error
ERFIX	Disk requires FIXUP
ERFTL	Filename too long
ERIFC	Invalid character in filename
ERLAB	Disk label does not match specified one
ERNAD	Non-directory name in pathname
ERPWL	Device write error

Convert a Time to Milliseconds *(library routine)*

?MSEC

Accepts a time in the form hours/minutes/seconds and returns a single 32-bit number representing the equivalent number of milliseconds. All inputs will be range checked; i.e., the hours must range from 0 to 23, and both the minutes and seconds must range from 0 to 59.

Inputs

AC	Contents
AC0	Seconds
AC1	Minutes
AC2	Hours

Outputs

AC	Contents
AC0	High order 16 bits of the time in milliseconds
AC1	Low order 16 bits of the time in milliseconds

Options

None

Errors

Mnemonic	Meaning
ERANG	Input out of range

?MSEG Map a Memory Segment

Maps all or part of a specified local or global segment* into the program logical address space, as specified by the ?MSEG packet. See Figure 12.10. The program must be attached to the specified segment before mapping is attempted.

The segment number being mapped is specified in packet word ?MSSN. Packet word ?MSSP specifies the starting page number within that segment. The program logical page number at which to start mapping is specified by packet word ?MSPB. Mapping continues through sequential segment and process pages, until the total number of requested segment pages (specified in packet word ?MSNB), have been mapped.

If the WP option is used, the mapped pages will be write protected. This means that the calling process will encounter a write protect trap if it attempts to modify the contents of the newly mapped area.

The old contents of those portions of user logical address space which are mapped to a new segment become inaccessible unless remapped.

User mapped segments are not swapped in or out by the ?EXEC or ?RETURN calls.

**A segment is an area of memory consisting of 1 to ?MXSP pages (1K word blocks). User created segments are identified and referenced by means of a global segment number assigned when the segment is created. See ?CSEG.*

Inputs

AC	Contents
AC2	Packet address

Outputs

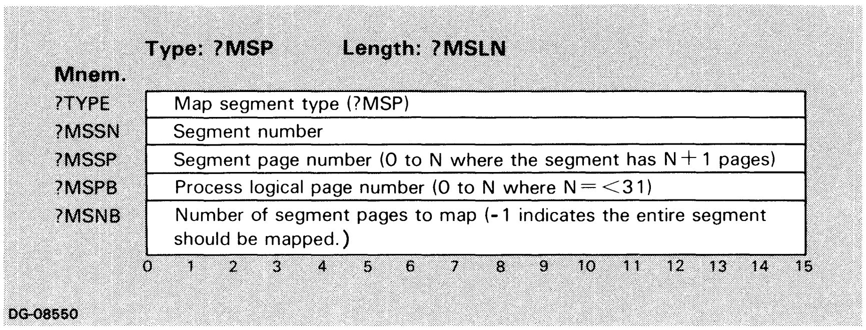
None

Options

Mnemonic	Meaning
WP	Write protect the pages mapped

Errors

Mnemonic	Meaning
ERIMA	Segment map area is not within 0-31
ERMLS	Request is longer than segment
ERSDE	Segment does not exist
ERSNA	Segment not attached



Type: ?MSP Length: ?MSLN

Mnem.	
?TYPE	Map segment type (?MSP)
?MSSN	Segment number
?MSSP	Segment page number (0 to N where the segment has N+1 pages)
?MSPB	Process logical page number (0 to N where N=<31)
?MSNB	Number of segment pages to map (-1 indicates the entire segment should be mapped.)

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
DG-08550

Figure 12.10 Map segment packet

?MYID Get Task or Process Identity

Places the calling task's identifier and priority in two accumulators. The PRC option causes process identifier and priority to be returned.

Inputs

None

Outputs

AC	Contents
AC0	Task priority Option: PRC: process priority
AC2	Task identifier Option: PRC: process identifier

Options

Mnemonic	Meaning
PRC	AC0, AC2 return process information

Errors

None

Get Obituaries *(IPC Server call)*

?OBITS

Returns a list of ports representing connections broken by the customer, but not yet broken by the server. (The customer terminated or issued an ?EXEC or a ?CLEAR call, but the server issued neither a ?CLEAR on that customer port nor a ?PURGE.)

This call allows a server to determine any broken connections before issuing a ?RCVA.

The server specifies the address and length of a buffer for receiving the list of broken connections from ?OBITS. If the buffer is too small for the entire list, it is filled and no error is returned.

The server can ?CLEAR the port numbers returned in the buffer and reissue ?OBITS to determine if there are additional broken connections.

A broken connection reported by ?OBITS is reported on all subsequent ?OBITS calls, unless that port is ?CLEARed by the server.

If the CK option is not used, ?OBITS waits until it has an obituary to report. If the CK option is used and there are no broken connections at the time of the call, the call immediately returns error ERNBC, *No broken connections.*

Inputs

AC	Contents
AC0	Server number
AC1	Word length of receive buffer
AC2	Word address of buffer to receive list of broken connections

Outputs

AC	Contents
AC1	Number of ports actually contained in buffer

Options

Mnemonic	Meaning
CK	Don't wait

Errors

Mnemonic	Meaning
ERSRV	Invalid server
ERSRN	Server name has been removed and server has no current connections
ERNBC	No broken connections (CK option only)

Open an I/O Channel

?OPEN

Returns an I/O channel to a specified device or disk file.

The pathname must be terminated by a null byte. The system returns a channel number to you in an accumulator. When a file is opened, the file pointer is set to zero (the first byte of the file), unless the AP option is used as detailed below.

File element size is discussed in Chapter 3.

When you allocate new blocks to a disk file, the system writes zeroes to all bytes in these blocks, unless the NZ option is used.

If you open a channel to a character output device, the system sends a Form Feed character (14_8) to it, unless the AP option is used.

For a list of file types, see Table 12.1.

Inputs

AC	Contents
AC0	Byte pointer to pathname
AC1	Options: CR: File type DE: File type UC: File type
AC2	Options: CR: File element size DE: File element size UC: File element size

Outputs

AC	Contents
AC0	Channel number.

Options

Mnemonic	Meaning
AP	For files, opens for appending; sets the file pointer to the end of the file. For character output devices, suppresses the sending of a Form Feed character.
CR	If the file does not exist, creates it.
DE	Deletes any existing copy of the file, then creates it.
EX	Exclusive access: no other program may use the file while this channel is open (gives an error return with code EREOP if the file is already in use.)
NZ	Don't zero new elements on allocation.
UC	Unconditionally creates the file (gives an error return if the file already exists).

NOTE: *If you specify either the DE or UC option, the searchlist is not used in resolving the file. If the file does not exist in the working directory, it is created there.*

Errors

Mnemonic	Meaning
ERDOL	Device off line
EREOP	File in use (EX option only), or file already ?OPENed with EX option
ERFDE	File does not exist
ERFIL	Device read error
ERFTL	Filename too long
ERIFC	Invalid character in filename
ERIOO	Illogical option combination
ERNAD	Non-directory name in pathname
ERNAE	File already exists (UC option only)
ERNMC	No more channels
ERPRM	Permanent file: cannot be deleted
ERPWL	Device write error

Load an Overlay (*library routine*)

?OVLOD

Checks to see if the specified overlay is currently in its node. If it is not, the overlay is loaded into the node.

If the node is occupied by another overlay, the calling task is pended until the node becomes available.

Inputs

AC	Contents
AC0	Overlay descriptor

NOTE: *For information about the format of overlay descriptors, see Appendix F, "Using Overlays."*

Outputs

AC	Contents
AC1	Base address of overlay start

Options

None

Errors

Mnemonic	Meaning
EROVN	Invalid overlay descriptor

?OVREL Release an Overlay (*library routine*)

Releases control of the specified overlay, and unpends any pended tasks awaiting that node.

If several tasks are pended awaiting different overlays for the node, the system selects one by the same selection method used by the ?UNPEND call.

Inputs

AC	Contents
AC	Overlay descriptor

NOTE: *For information about the format of overlay descriptors, see Appendix F, "Using Overlays."*

Outputs

None

Options

None

Errors

Mnemonic	Meaning
EROVN	Invalid overlay descriptor
EROVC	Specified overlay is not currently loaded

Suspend a Task

Causes the calling task to become suspended from execution until a specified event occurs. The event is defined by a 16-bit number which may be used by another task in an ?UNPEND call. Event numbers must be greater than or equal to 0 and less than or equal to ?EVMAX. When execution resumes, the system passes a message word from the task which executed the ?UNPEND.

The calling task may also resume execution in response to an ?UNPEND ID call or after a timeout interval elapses. The length of the interval in milliseconds is specified as a 32-bit number in two accumulators. You may also request the system default timeout interval (about one minute) by setting both accumulators to 0.

NOTE: *A ?PEND with a timeout value of -1 will pend forever.*

If ?PEND is issued when task scheduling is disabled, ?PEND re-enables scheduling after blocking the calling task.

To suspend a task while interrupts are disabled, issue ?IPEND, rather than ?PEND. ?IPEND guarantees that the pend is internally queued before interrupts are reenabled.

A program may receive console interrupts by creating a task which pends until an event number equal to ?EVCH plus the channel number of the user's console keyboard occurs. This task is unpended when the user types the control sequence CTRL-C CTRL-A on the keyboard.

Inputs

AC	Contents
AC0	Event number
AC1	High order word of timeout duration
AC2	Low order word of timeout duration

Outputs

AC	Contents
AC0	Message word from ?UNPEND

Options

None

Errors

Mnemonic	Meaning
ERCIN	Console interrupt
EREVT	Invalid event number
ERTMO	Timeout interval has elapsed

Change Task or Process Priority

?PRI

Sets the value of the specified task's priority. Priorities may range from 0 to 255; lower values have higher priorities.

The PRC option allows process priority to be modified. Process priority has precedence over task priority. See discussion on scheduling in Chapters 4 and 8.

Inputs

AC	Contents
AC0	New task priority Option: PRC: process priority
AC2	Task identifier: zero means this task Options: PRC: process identifier; zero means this process.

Outputs

None

Options

Mnemonic	Meaning
PRC	Information in AC0 and AC2 pertains to process

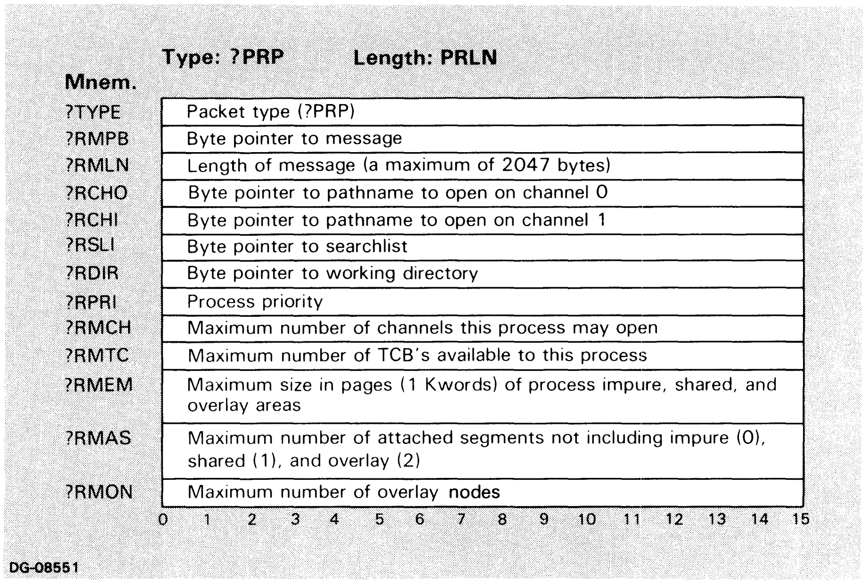
Errors

Mnemonic	Meaning
ERPRP	Invalid priority
ERTID	Invalid task identifier

?PROC Create a Process

Starts a new process with a separate address space which will execute in parallel with the calling process. The specified program file is executed at swap level 1.

?PROC requires a process definition packet whose format is described below. See Figure 12.11.



Type: ?PRP Length: PRLN

Mnem.	
?TYPE	Packet type (?PRP)
?RMPB	Byte pointer to message
?RMLN	Length of message (a maximum of 2047 bytes)
?RCHO	Byte pointer to pathname to open on channel 0
?RCHI	Byte pointer to pathname to open on channel 1
?RSLI	Byte pointer to searchlist
?RDIR	Byte pointer to working directory
?RPRI	Process priority
?RMCH	Maximum number of channels this process may open
?RMTC	Maximum number of TCB's available to this process
?RMEM	Maximum size in pages (1 Kwords) of process impure, shared, and overlay areas
?RMAS	Maximum number of attached segments not including impure (0), shared (1), and overlay (2)
?RMON	Maximum number of overlay nodes

Figure 12.11 Process definition packet

A setting of −1 for any of the packet fields specifies that the default value (that of the creating process) is to be used. There is no default value for ?RMLN (length of message) and ?RMBP (byte pointer to message). See Table 12.5.

If ?RCH0 or ?RCH1 is set to −1, ?INCH or ?OUCH for the new process will be the same as for the initiating process. In some cases (as when ?INCH is opened exclusively), this causes ?PROC to return an error without creating a new process.

A nonzero or a −1 (default) setting must be specified for the following fields): searchlist (?RSLI), working directory (?RDIR), number of tasks (?RMTC), memory size (?RMEM).

Setting a packet field to zero produces different effects depending on the field. See Table 12.5. In some cases a zero setting indicates that the particular feature is not desired.

Packet Field	Contents	Effect of 0 Setting	Effect of -1 Setting
?RMBP	Byte pointer to message	No message	Error
?RMLN	Length of message	No message	Error
?RCH0	Byte pointer to pathname, channel 0	Closed	Default value
?RCH1	Byte pointer to pathname, channel 1	Closed	Default value
?RSLI	Byte pointer to searchlist	Error	Default value
?RDIR	Byte pointer to working directory	Error	Default value
?RPRI	Process priority	Valid	Default value
?RMCH	Maximum number of channels	Valid	Default value
?RMTC	Maximum number of TCB's	Error	Default value
?RMEM	Maximum size of memory	Error	Default value
?RMAS	Maximum number of attached segments	Valid	Default value
?RMON	Maximum number of overlay nodes	Valid	Default value

Table 12.5 Process definition packet: effect of zero and minus one settings on each field.

Thus for example, if ?RCH0 or ?RCH1 is set to 0, no devices are opened by the system on channels 0 or 1, i.e., ?INCH or ?OUCH channels are not opened for the new process. Setting the message fields ?RMBP and ?RMLN to zero indicates that there is no message.

In the fields specifying priority, channels, segments, and overlay nodes (?RMPRI, ?RMCH, ?RMAS, ?RMON), a setting of zero is a valid value.

In the fields indicating searchlist, working directory, TCB's, and memory size (?RSLI, ?RDIR, ?RMTC, ?RMEM) a setting of zero results in an error.

The process priority (?RPRI) may range from 0 to 255. Lower values have higher priorities. Process priority is distinct from task priority and supersedes it. See discussion of scheduling in Chapter 4.

The value of the Process ID returned in AC2 may range between 1 and 65535. ID numbers are assigned in ascending order; when ID number 65535 has been assigned, the next ID to be assigned is 1 or the lowest numbered free ID available. A process ID number references the process in a variety of calls such as ?KILL, ?BLOCK, or ?INFO.

The size of process *impure* memory (segment 0), may be increased or decreased. See ?MEMI.

Inputs

AC	Contents
AC0	Byte pointer to pathname of program file
AC2	Address of packet

Outputs

AC	Contents
AC2	Process ID of new process

Options

Mnemonic	Meaning
DB	The process being created will be debugged

Errors

Mnemonic	Meaning
ERBTL	Message buffer is too long
ERDOL	Device off line
ERFDE	File does not exist
ERFIL	Device read error
ERFTL	File name too long
ERIFC	Invalid character in pathname
ERIFT	Invalid file type (not a program file)
ERIPT	Invalid packet type
ERIRB	Message buffer is too short
ERMEM	Insufficient memory to load the process
ERMPR	Invalid packet address
ERNAD	Non-directory name in pathname
ERPRP	Invalid priority
ERPWL	Device write error

Server Purge Request *(IPC Server call)*

?PURGE

Breaks the connection between a single server and all of its current customers.

Any outstanding ?RCV, ?SEND, and ?SD.R calls to or from this server terminate in error ERCBS, ***Connection broken by server***. If the server has previously issued a ?RMVE and has outstanding ?RCVA or ?OBITS calls, then the ?RCVA and ?OBITS calls are terminated with error ERSRN, ***Server name has been removed and server has no current connections***.

Inputs

AC	Contents
AC0	Server number

Outputs

None

Options

None

Errors

Mnemonic	Meaning
ERSRV	Invalid server

?RCV Receive Data Request *(IPC Server/Customer call)*

Allows caller to receive an IPC message from a specific sender process.

Before issuing ?RCV, the process must set up a packet in the format described in Figure 12.12.

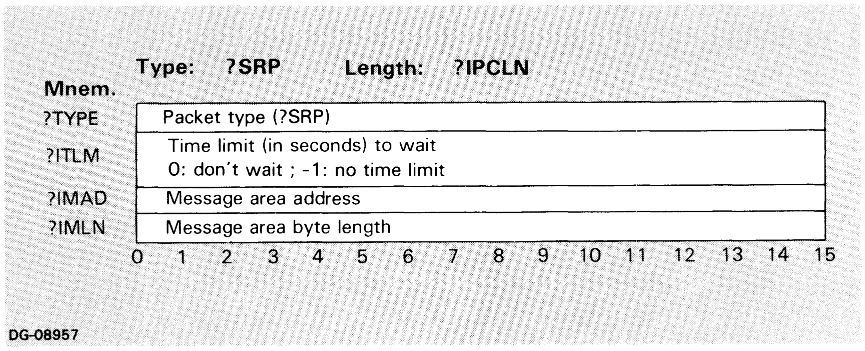
Offset ?ITLM in the packet specifies the maximum time in seconds during which ?RCV is suspended to await a matching ?SEND request. If this time limit is set to zero (?ITLM = 0, i.e., no waiting time), and no matching message is posted to the calling process, an error is returned. Similarly, when the time limit is exceeded, ?RCV returns in error.

?ITLM functions independently of the NP option which is used with ?AWAIT.

Offsets ?IMAD and ?IMLN indicate the receive buffer byte address. This address can be in any area in the user address space, or in a user-defined segment currently mapped to the user address space, provided that area is neither write- nor validity protected. Offset ?IMLN indicates the size in bytes of the receive buffer. If the buffer area is insufficient for the message, an error is returned, and no part of the message is transmitted. However, the actual length of the message is returned in AC1, enabling the receiving process to reissue ?RCV after modifying the size of the receive buffer area.

The sender then continues to wait for the receiver to receive the message properly, until the sender's time limit (?ITLM) is exceeded.

?RCV calls are serviced in first-in, first-out fashion: when more than one task issues a ?RCV on messages from the same port, the next message sent from that port is received by the task which first posted the ?RCV.



Type: ?SRP Length: ?IPCLN

Mnem.	
?TYPE	Packet type (?SRP)
?ITLM	Time limit (in seconds) to wait 0: don't wait ; -1: no time limit
?IMAD	Message area address
?IMLN	Message area byte length

DG-08957

Figure 12.12 Receive packet

Inputs

AC	Contents
AC0	Port number used to communicate with the destination process
AC2	Packet address

Outputs

AC	Contents
AC1	Number of bytes actually received (or message length if message area byte length is too small)

Options

Mnemonic	Meaning
NP	Nonpended

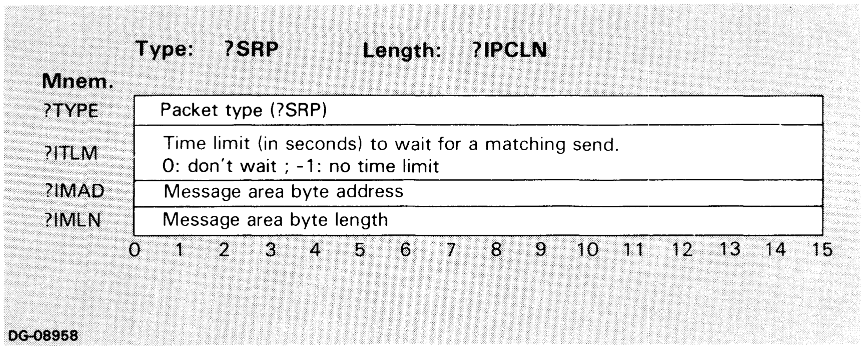
Errors

Mnemonic	Meaning
ERCBC	Connection broken by customer
ERCBS	Connection broken by server
ERPOR	Invalid port
ERSMB	Receive buffer too small
ERTMO	Pend time out (time limit expired)

?RCVA Receive Any Request *(IPC Server call)*

A "receive-any" function which returns a message queued to the calling server process from any customer process.

Before issuing this call, the server process must set up a packet in the format described in Figure 12.13.



Type: ?SRP **Length:** ?IPCLN

Mnem.	0–15
?TYPE	Packet type (?SRP)
?ITLM	Time limit (in seconds) to wait for a matching send. 0: don't wait ; -1: no time limit
?IMAD	Message area byte address
?IMLN	Message area byte length

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
DG-08958

Figure 12.13 Server receive packet

Offset ?ITLM in the packet specifies the maximum time in seconds during which ?RCVA is suspended to await any request. If this time limit is set to zero (?ITLM = 0) and no matching message is posted to the calling process, an error is returned.

?ITLM functions independently of the NP option which functions as usual in conjunction with the ?AWAIT call.

Packet words ?IMLN and ?IMAD set up a buffer byte size and a byte address to receive the message. The receive buffer address can be in any area within the user logical address space, or in a user-defined segment currently mapped to the calling process, provided that this area is neither write nor validity protected.

If the receive buffer area is insufficient for the message, an error is returned and no part of the message is transmitted. However, the actual length of the message is returned in AC1, and the port used by the sender is returned in AC2. This enables the receiving process either to reissue ?RCVA, or to issue a ?RCV call instead, after modifying its receive buffer size. Within the timeout limits it has specified in ?ITLM, the sender continues to wait for the receiver to receive the message properly.

The port number returned in AC0 is a value returned by the system, and internally associated with the connection between the process originating the message and the server. This port number can be used by the server process to send messages back to that specific customer process.

To prevent ?RCVA from intercepting messages for which a specific ?RCV is posted, ?RCV calls have precedence over ?RCVA calls. If a server has a task pended on a ?RCV from a specific port, and another task pended on a ?RCVA, a message from the specified sender port is received by ?RCV and not by ?RCVA. This is true regardless which call was issued first.

Inputs

AC	Contents
AC0	Server number
AC2	Packet address

Outputs

AC	Contents
AC0	Actual port used by sender
AC1	Number of bytes received (or length of entire message, if receive buffer is too short)
AC2	Actual port used by sender (in case of error ERSMB only)

Options

Mnemonic	Meaning
NP	Nonpended

Errors

Mnemonic	Meaning
ERSMB	Receive buffer too small (for this error, the actual port used by the sender is returned in AC2, and the message length is returned in AC1).
ERSRN	Server name has been removed and server has no current connections.
ERSRV	Invalid server.
ERTMO	Pend time out (time limit expired).

?RDMEM Read Memory *(Debugger call)*

Examines memory locations in the program being debugged, starting from the location specified in AC0, and reading the number of words specified in AC1. The contents of these locations can then be modified via the ?WRMEM system call.

Before issuing ?RDMEM, the caller must prepare a buffer area to receive the information read. The buffer area can be anywhere within the caller's logical address space, or within a user-defined segment currently mapped to the calling process, provided that this area is neither write nor validity protected.

If the buffer area is insufficient to hold the data read, the buffer is filled and no error is returned.

Inputs

AC	Contents
AC0	Address in target program for start of read
AC1	Length in words of data to read
AC2	Address of buffer to receive the data read

Outputs

None

Options

None

Errors

Mnemonic	Meaning
ERBTL	Buffer extends out of user space
ERNDB	No target program defined
ERTAD	Invalid target program address

Read Task State *(Debugger call)*

?RDST

Returns information about the task whose identifier is specified in AC0. Task ID numbers are retrieved with ?GIDS.

Before issuing ?RDST, the caller must set up an area to receive a packet of the format described in Figure 12.14. The packet address can be in any area of the caller's logical address space, or in a user-defined segment currently mapped to the calling process, provided that this area is neither write nor validity protected.

In general, the contents of the packet words are returned as unsigned octal integers; the exceptions are as follows:

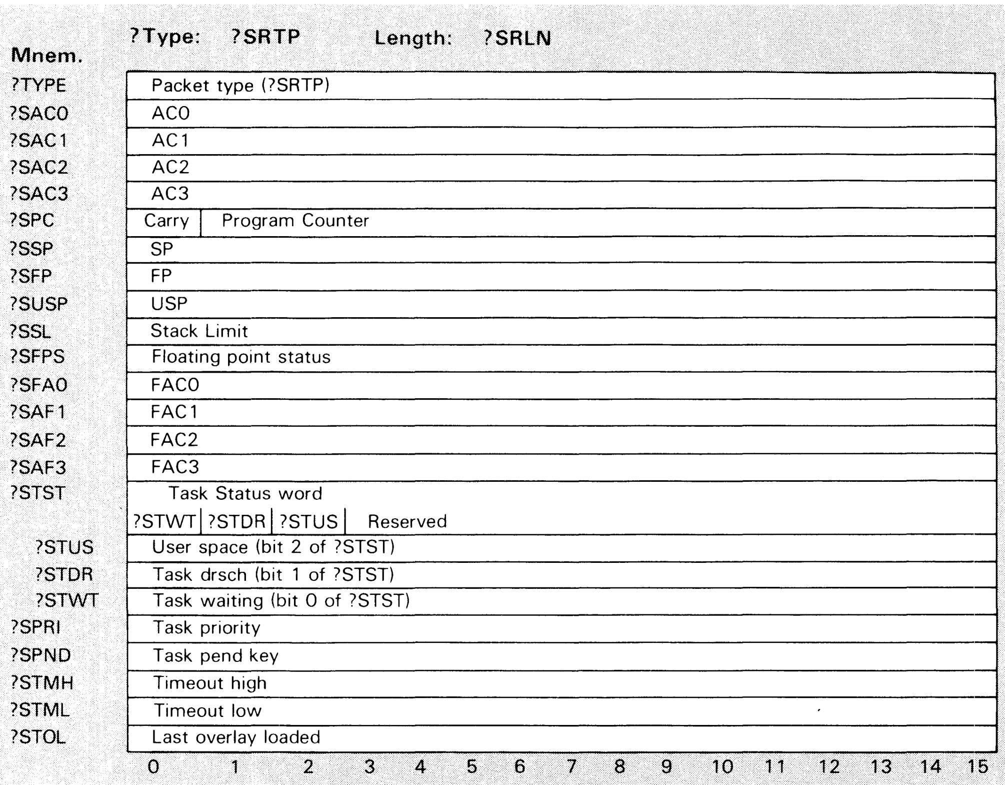
- the Carry Register is displayed as bit 0 of the program counter (?SPC);
- user space, task drsch and task waiting are expressed as single bits of the task status word (?STST);
- the floating register contents (FAC0 through FAC3) are returned in double precision floating-point format.

Packet words ?SAC0 through ?SAC3 contain the values of accumulators 0 through 3. The locations of the hardware stack pointer and hardware frame pointer are contained in words ?SSP and ?SFP. Packet words ?SUSP, ?SSL, and ?SFPS contain the values of the Unique Storage Position, the stack limit and the floating point status word, an octal number in the range of 0 - 377. All task state information up to this point (i.e., packet words ?SAC0 through ?SAF3), is modifiable by the debugger program via the ?WRST system call.

The information in the last ten packet words (?STST through ?STOL) is protected from modification and is available as read-only information for the debugger. The task status word (?STST) contains three bits as follows:

- ?STUS (user space, bit 2 of ?STST): when set, this bit indicates that the task is currently running user code rather than executing in system space;
- ?STDR (task drsch, bit 1 of ?STST): when this bit is set, the task did not issue a ?DRSCH system call;
- ?STWT (task waiting, bit 0 of ?STST): when set, this bit indicates that the task is waiting for a nonpended system call to complete.

The packet words following ?STST specify task priority, pend key, 32-bit timeout, and the descriptor for the last overlay loaded.



?Type: ?SRTP Length: ?SRLN

Mnem.	
?TYPE	Packet type (?SRTP)
?SACO	ACO
?SAC1	AC1
?SAC2	AC2
?SAC3	AC3
?SPC	Carry | Program Counter
?SSP	SP
?SFP	FP
?SUSP	USP
?SSL	Stack Limit
?SFPS	Floating point status
?SFAO	FACO
?SAF1	FAC1
?SAF2	FAC2
?SAF3	FAC3
?STST	Task Status word ?STWT | ?STDR | ?STUS | Reserved
?STUS	User space (bit 2 of ?STST)
?STDR	Task drsch (bit 1 of ?STST)
?STWT	Task waiting (bit 0 of ?STST)
?SPRI	Task priority
?SPND	Task pend key
?STMH	Timeout high
?STML	Timeout low
?STOL	Last overlay loaded

DG-08959

Figure 12.14 Read task status packet

Inputs

AC	Contents
ACO	Identifier of task to examine
AC2	Address of packet

Outputs

None

Options

None

Errors

Mnemonic	Meaning
ERIPT	Invalid packet type
ERMPR	Invalid packet address
ERTID	Invalid task identifier

Read Data From a Device or File

?READ

Reads one or more bytes from the specified I/O channel, or one or more bytes from a device to a memory segment.

?READ may operate in *dynamic, data-sensitive,* or *extended* mode. For dynamic input, you specify the number of bytes to be read, as well as the address at which to store the data.

If you are reading from a disk, you can improve the efficiency of your program by transferring entire disk blocks. This method eliminates system overhead for buffering. To execute block-aligned transfers you must

- set the file pointer to a multiple of 512 before the transfer;
- specify a multiple of 512 bytes to read;
- specify a buffer which is word aligned in your address space.

For *data-sensitive* I/O (DS option), you specify a maximum number of bytes, and reading proceeds until a *delimiter* is read. Delimiters may be either the default delimiters New-Line (12_8), Carriage Return (15_8), Form Feed (14_8), or null (00_8), or any characters specified by a user *delimiter table.* (See ?SCHS for a discussion of delimiter tables.) The delimiter is counted in the returned number of bytes read, and the delimiter character actually appears in the buffer.

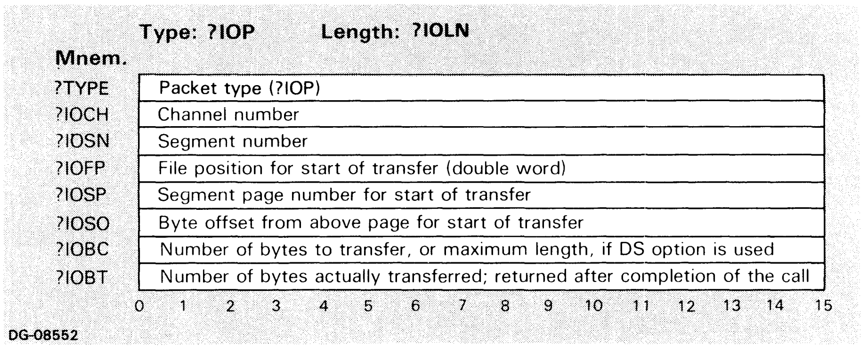
NOTE: *When you execute a data-sensitive ?READ, you may encounter the end of file before finding a delimiter. Similarly, on a dynamic ?READ there may not be as many bytes left as you requested. In either case, the call will take the error return, but the data will be read and AC2 will contain the number of bytes read, including the delimiter. The delimiter character will also be included in the buffer.*

Extended I/O is selected by the PKT option. In this case you specify the address of a packet whose format is described below in Figure 12.15.

This form of reading transfers the data to a specified memory segment. The memory segment need not be mapped to the user's address space. To determine the segment address for the start of the transfer, the packet specifies a page number, as well as a byte offset.

Extended I/O may be either dynamic or data sensitive. It is *not* allowed on character devices. After the end of the transfer, two values are returned: word ?IOBT of the extended I/O packet returns the number of bytes transferred; double word ?IOFP is updated to reflect the current position in the file.

NOTE: *The block transfer optimization also applies to extended (PKT option) reads.*



Type: ?IOP Length: ?IOLN

Mnem.	
?TYPE	Packet type (?IOP)
?IOCH	Channel number
?IOSN	Segment number
?IOFP	File position for start of transfer (double word)
?IOSP	Segment page number for start of transfer
?IOSO	Byte offset from above page for start of transfer
?IOBC	Number of bytes to transfer, or maximum length, if DS option is used
?IOBT	Number of bytes actually transferred; returned after completion of the call

DG-08552

Figure 12.15 Extended I/O packet

Inputs

AC	Contents
AC0	Channel number
AC1	Byte pointer to buffer to receive data
AC2	Byte count (dynamic) Options: DS: maximum byte count PKT: packet address

Outputs

AC	Contents
AC2	Actual number of bytes read, including terminator Option: NP: system task ID

Options

Mnemonic	Meaning
DS	Data-sensitive read
FL	For character devices: causes any characters currently held in the system to be discarded. (Flush type ahead.)
IX	Only checked if the DS option is selected: ignore input after maximum byte count is reached, or until a delimiter is typed. The default returns an error, ERLTL if the byte count is exceeded.
NP	Nonpended call. Returns system task ID in AC2.
PKT	Extended read from a device to an arbitrary memory segment. Not allowed on character device.

Errors

Mnemonic	Meaning
ERBTL	Buffer is too long
EREOF	End of file encountered
ERICN	Invalid channel number
ERIRB	Buffer too short
ERLTL	Line too long: too many bytes without a delimiter (DS option only)
ERNOT	No free task control blocks (NP option only)
ERRAD	Read access denied

?RENAME Rename a File

Give a new pathname to a disk file. The new and old pathnames must both be *on the same disk device.* If a file with the new pathname already exists, the call gives an error return, unless the DE option is specified. The file must not be open. If the last filename in the pathname pointed to by AC0 is a link, the link itself is renamed, not the link resolution.

NOTE: *If the new pathname points to a different directory, you can effectively ''move'' the file into the new directory.*

If the specified new pathname is not fully qualified, and the file is not found in the working directory, it is created there. The searchlist is not scanned.

Inputs

AC	Contents
AC0	Byte pointer to current pathname
AC1	Byte pointer to new pathname

Outputs

None

Options

Mnemonic	Meaning
DE	If the new filename exists, delete that file before renaming.

Errors

Mnemonic	Meaning
ERDOL	Device off line
ERFDE	File does not exist
ERFIL	Device read error
ERFTL	Filename too long
ERIFC	Invalid character in filename
ERIFT	Illegal file type (attempt to rename a device)
ERNAD	Non-directory name in pathname
ERPRM	Permanent file: cannot be renamed
ERPWL	Device write error
ERREN	Attempt to rename across devices or to rename a root directory or an open file

Close Multiple I/O Channels

?RESET

Closes I/O channels, as specified by a 16-bit mask that you place in AC0. No error is produced if you attempt to close a channel that is already closed. Only the first 16 channels may be closed with ?RESET.

NOTE: *Channels ?INCH and ?OUCH are set up by the system for standard input and output; therefore, it is generally convenient for you to keep them open.*

Inputs

AC	Contents
AC0	Any bit (0 - 15) set to 1 causes the corresponding channel to be closed

Outputs

None

Options

None

Errors

Mnemonic	Meaning
ERDOL	Device off line
ERFIL	Device read error
ERPWL	Device write error

?RETURN Return to the Previous Program Level

Terminates the *process,* if the returning program was running at *swap level* 1, or resumes execution of the *parent* program if the returning program runs from a level *other* than 1. You may cause the parent to take the error return from its ?EXEC call. All I/O channels are closed: however, the parent program will have the same I/O status that it did when it performed its ?EXEC. Segments are implicitly detached from the returning program. You may pass an error code and/or a message of up to 2,047 bytes to the parent program.

When execution of the parent program is resumed, that program's segment environment is restored.

If a ?RETURN is executed from a program executing at swap level 1 in the *initial* process, the system is shut down.

?RETURN never takes the error return. If the parent program cannot be resumed, an error code is passed to the "grandparent" program, i.e., two program levels previous instead of one.

If you specify the BK option, ?RETURN creates a *break file* from the current program in the system working directory (:?SYSDIR). The name of the break file is composed of a question mark (?) followed by seven digits and a .BRK extension, thus:
?xxxxxxx.BRK

The first six digits following the ? represent the PID (process identifier) of the process within which the calling program is running. The seventh digit represents the program execution level (swap level) of the calling program. The break file is written into the system working directory; any existing file of the same name is overwritten.

The current memory image of the terminating program as well as information about the program state, task states, user overlays in use, attached segments, and all open files is written to the break file.

Inputs

AC	Contents
AC0	Error code to return to parent program; if zero, the parent will take the normal return from its ?EXEC call
AC1	Byte pointer to message (if AC2 is nonzero)
AC2	Message length in bytes

Outputs

None

Options

Mnemonic	Meaning
BK	Save program state in a break file.

Errors

None (See text.)

?RMVE Server Process Declaration Removal *(IPC Server call)*

Removes the current process from the named association declared in the ?DCLR call and invalidates the server number identified with it.

?RMVE has no effect on any existing connections. Its intent is to prevent the establishing of new connections by customers looking up this server via ?LKUP.

If there are no current connections to this server (i.e., if ?CLEAR or ?PURGE calls were previously issued, or if no process has looked up this server), any outstanding ?RCVA or ?OBITS calls to this server are terminated in error *ERSRN, Server name has been removed and server has no current connections.*

Inputs

AC	Contents
AC0	Server number

Outputs

None

Options

None

Errors

Mnemonic	Meaning
ERSRV	Invalid server

Set Device Characteristics

?SCHAR

Sets the characteristics of the specified device.

The device name must be terminated by a null byte. The specified device must be a character device. The characteristics are summarized in Table 12.6.

Inputs

AC	Contents
AC0	Byte pointer to device name
AC1	Options: HC terminal's characteristics word LL characters per line PG lines per page

Outputs

None

Options

Mnemonic	Meaning
CH*	Input a channel number instead of a byte pointer to a pathname in AC0.
HC	Set terminal's hardware characteristics in AC1 (for hardware with programmable characteristics only).
LL	Set the number of characters per line to the value in AC1.
PG	Set the number of lines per page to the value in AC1.
RS	Reset characteristics to their value at boot time. (Does not reset lines per page or characters per line.)

**If the CH option is used, characteristics are set for the device opened on the channel specified in AC0. This does not render the characteristics channel specific: an error will be returned if the channel is not open on a character device.*

Set On	Mnemonic	Affects	Meaning when 1
Input/ Output	?CBIN	Both	Binary mode: disables all special control characters; passes all characters exactly as received (8 bits).
Input	?CECH	Output	Echo mode: echoes all typed characters although some receive special handling as described in text.
Input	?CEMM	Output	Echo characters exactly as input: turns off echoing of control characters as ↑x.
Input	?CESC	Input	Escape mode: handles Escape (33_8) the same as CTRL-C CTRL-A.
Input	?CICC	Both	Ignore control characters except delimiters and characters interpreted by the system.
Output	?CLST	Output	List mode: echoes Form-Feeds (014_8)as "^L" to prevent them from erasing CRT screen.
Input/ Output	?CNAS	Both	Non-ANSII-standard console: supports terminals using older standard for control characters by converting Carriage Returns (015_8) into New-Lines (012_8), and vice versa, on input; on output, converts New-Line to Carriage Return, followed by New-Line, followed by null.
Input	?CNED	Output	Do not echo delimiters.
Output	?CST	Output	Simulates tabs: converts all tab characters (011_8) to the appropriate number of spaces; cursor moves to the beginning of the next 8-character tab column.
Output	?CUCO	Output	Convert to uppercase on output.
Output	?C605	Both	6052, 6053 or similar device: uses cursor movement characters to echo Rubout and CTRL-U by erasing characters from the screen. The two characters following a 37_8 on input and a 20_8 on output will be passed through uninterpreted.
Input/ Output	?C8BT	Input	8-bit characters; the default is to mask all input characters to 7 bits, unless in binary mode.

Table 12.6 Console characteristics

NOTE: *Not all characteristics have meaning for all devices. (See Chapter 9, "I/O Device Management".)*

If you specify the LL option and set the line length to −1, neither line length checking nor automatic wrap-around will be done.

The HC option is for hardware with programmable characteristics only. These characteristics are: number of stop bits, parity, code level, baud rate for Asynchronous/Synchronous Line Multiplexors (ASLM's), and clock selection for Asynchronous Line Multiplexors (ALM's).

If the HC option is selected, AC1 will contain the terminal's characteristics in the following format, as summarized in Tables 12.7 through 12.9. Unused bits must be zero.

Number of Stop Bits	Parameter	Parity	Parameter	Code Level	Parameter
1	?C1S	None	?CNPR	5 bits	?C5BC
1.5	?C15S	Odd	?CODD	6 bits	?C6BC
2	?C2S	Even	?CEVN	7 bits	?C7BC
				8 bits	?C8BC

Table 12.7 Programmable hardware characteristics (?SCHAR with HC option)

Baud Rate for ASLM	Parameter
50	?C0050
75	?C0075
110	?C0110
134.5	?C1345
150	?C0150
300	?C0300
600	?C0600
1200	?C1200
1800	?C1800
2000	?C2000
2400	?C2400
3600	?C3600
4800	?C4800
7200	?C7200
9600	?C9600
19.2K	?C192K

Table 12.8 Baud rate for asynchronous/synchronous line multiplexors (ASLM boards)

ASLM 4336-S, 4336-AS characteristics may be changed at any time.

Clock	Parameter
0	?CCK0
1	?CCK1
2	?CCK2
3	?CCK3

Table 12.9 Clock selection for asynchronous line multiplexors (ALM boards)

Errors

Mnemonic	Meaning
ERDOL	Device off line
ERFDE	File does not exist
ERFIL	Device read error
ERFTL	Filename too long
ERICH	Invalid characteristics
ERIFC	Invalid character in filename
ERIFT	Not a character device
ERNAD	Non-directory name in pathname
ERPWL	Device write error

Set Channel Specifications ?SCHS

Changes a number of input characteristics on a per channel basis.

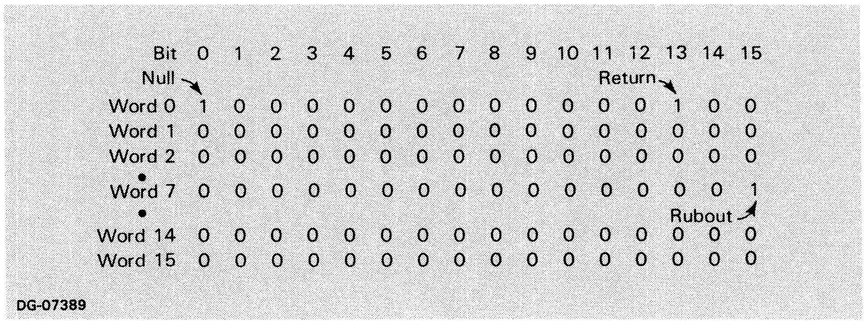
NOTE: *This works for disk I/O as well as character I/O.*

Setting AC2 to a value other than -1 changes the line delimiters used by ?READ and ?WRITE: if you set AC2 to 0, the default delimiters New-line (12_8), Carriage Return (15_8), Form Feed (14_8), or null (00_8) are used. If you wish to use a different set of characters as delimiters, set AC2 with the value of a word pointer to a *delimiter table*.

A *delimiter table* consists of 16 consecutive 16-bit words which form a table of 256 bits. Each bit in this table represents an eight-bit character. Reading from left to right, the first bit (bit 0) of the first word represents the null character (000). Likewise, the last bit (bit 15) of the last word in this table represents the binary character 377.

Setting any bit in this table to 1 indicates that the character represented by this bit will be a delimiter in data-sensitive records transmitted over that channel. Figure 12.16 shows a delimiter table with bits set to make null (000), Carriage Return (15_8), and Rubout (177_8) data-sensitive record delimiters.

The specifications established by this call are passed to descendants on an ?EXEC call, if the channel itself is passed.



Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	Null													Return		
Word 0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
Word 1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Word 2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
•																
Word 7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
•																Rubout
Word 14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Word 15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

DG-07389

Figure 12.16 Sample delimiter table

Inputs

AC	Contents
AC0	Channel number
AC1	Specifications
AC2	Word pointer to delimiter table 0 ⟶ use default delimiters -1 ⟶ don't change delimiters

Outputs

None

Options

None

Errors

None

Specifications

Mnemonic	Meaning
?CUCI	Uppercase input: convert all input characters to uppercase before putting in user buffer (does not affect the echo); only for character I/O.

Set the System Calendar *(library routine)*

?SDAY

Allows a program to adjust the system's internal calendar during execution. The routine accepts a date in day, month and year format and sets the system calendar to these values. Note that the year must be expressed as an offset from a base of 1900.

Inputs

AC	Contents
AC0	Day (range 1-31_{10})
AC1	Month (range (1-12_{10})
AC2	Year (minus 1900; result expressed in octal)

Outputs

None

Errors

Mnemonic	Meaning
ERANG	Range error

?SD.R Send/Receive Data Request *(IPC Server/Customer call)*

Performs a ?SEND, and, if successful, follows it immediately with a ?RCV.

?SD.R is a system defined ?SEND/?RCV pair. It does a ?SEND of ?ISDL bytes from ?ISAD followed by a ?RCV of a maximum of ?IRLN bytes into the area specified by ?IRLN.

Before issuing ?SD.R, the process must define the message and set up a receive buffer for the receiver's reply. In addition, the process must set up the ?SD.R packet whose format is described in Figure 12.17.

The message is sent and the reply received over the same port.

During the message transmission, the system suspends the ?SD.R calling task for the number of seconds specified in offset ?ITLM of the packet to await the matching receive call. If this time limit is exceeded, the call returns in error. Similarly, the receive portion of the ?SD.R call is suspended for the duration of ?ITLM. This time limit is identical for the ?SEND and ?RCV sequences of the call.

?ITLM functions independently of the NP option which functions as usual via the ?AWAIT call.

Packet words ?IRAD and ?IRLN indicate the buffer byte address and byte length for the reply received by the calling process.

The buffer address can be in any part of the user address space or in a user-defined segment mapped to the user address space, provided that area is neither write nor validity protected.

If the receive buffer area is insufficient to hold the message, an error is returned and no bytes are transmitted. However, the actual length of the message is returned in AC1. This enables the receiver to reissue ?RCV after modifying the size of the receive buffer.

Type: ?S__RP Length: ?ISRLN

Mnem.	
?TYPE	Packet type (?S_RP)
?ITLM	Time limit (in seconds) to wait 0: don't wait ; -1: no time limit
?ISAD	Message byte address
?ISDL	Message byte length
?IRAD	Receive area byte address
?IRLN	Receive area byte address

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

DG-08960

Figure 12.17 Send/receive packet

Inputs

AC	Contents
AC0	Port number used by caller to communicate with destination process
AC2	Packet address

Outputs

AC	Contents
AC1	Number of bytes received (or actual message length if receive area byte length is too small).

Options

Mnemonic	Meaning
NP	Nonpended

Errors

Mnemonic	Meaning
ERCBC	Connection broken by customer
ERCBS	Connection broken by server
ERPOR	Invalid port
ERSMB	Receive buffer too small
ERTMO	Pend time out (time limit expired)

?SEND Send Data Request *(IPC Server/Customer call)*

Sends a message from the caller to a designated server or customer process.

The port number of the receiving process must be input in AC0. (This port number is determined by means of a ?LKUP call if the sending process is a customer; if the sending process is a server the customer port number is returned to the server after the first ?RCVA.)

Before issuing ?SEND, the calling process must define the message send packet whose format is described in Figure 12.18 below.

Offset ?ITLM in the sender packet specifies the maximum time in seconds during which ?SEND is suspended to await a matching receive signal (?RCVA, or ?RCV) to be posted. If this time is exceeded, the system returns an error.

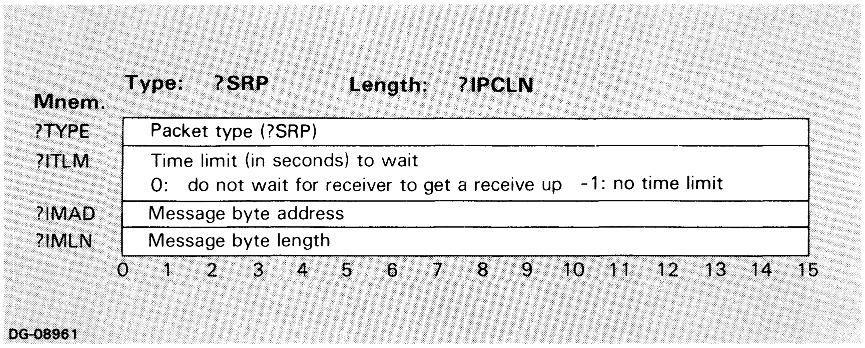
If the maximum time limit is set to zero (?ITLM = 0, i.e., no waiting time) and no matching receive request is posted to the calling process, the ?SEND request returns an error.

?ITLM is independent of the NP option which functions as usual via the ?AWAIT call.

?SEND calls are serviced in a first-in first-out fashion. If several tasks in a program are pended on a ?SEND to the same port, the message received by the target program when it posts a ?RCV is that sent by the first task issuing the ?SEND call.

Similarly, if ?SEND calls to the same server are issued by several tasks from different programs, the server on posting a ?RCVA receives the message sent by the first customer program to issue the ?SEND.

If insufficient space is available to transfer the message after a ?SEND request has locked onto the target process' receive request, an error is propagated to the receiver only. (See ?RCV or ?RCVA.) Within its specified time limit (?ITLM), the sender then waits for a receiver as if it had never found the waiting receiver with the insufficient buffer.



Type: ?SRP Length: ?IPCLN

Mnem.	
?TYPE	Packet type (?SRP)
?ITLM	Time limit (in seconds) to wait 0: do not wait for receiver to get a receive up -1: no time limit
?IMAD	Message byte address
?IMLN	Message byte length

DG-08961

Figure 12.18 Send request packet

Inputs

AC	Contents
AC0	Port number of destination process
AC2	Packet address

Outputs

None

Options

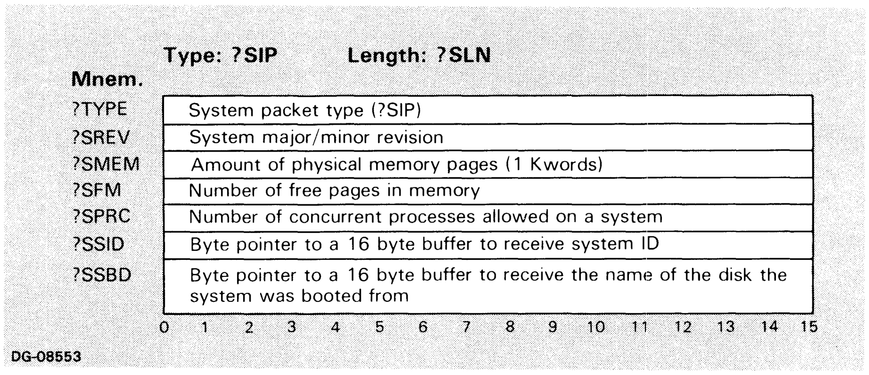
Mnemonic	Meaning
NP	Nonpended

Errors

Mnemonic	Meaning
ERCBC	Connection broken by customer
ERCBS	Connection broken by server
ERPOR	Invalid port
ERTMO	Pend time out (time limit expired)

?SINFO Get Information About the System

Returns a packet containing information about the system's revision, memory, and allowable number of concurrent processes. The packet format appears below in Figure 12.19.



Type: ?SIP Length: ?SLN

Mnem.	0–15
?TYPE	System packet type (?SIP)
?SREV	System major/minor revision
?SMEM	Amount of physical memory pages (1 Kwords)
?SFM	Number of free pages in memory
?SPRC	Number of concurrent processes allowed on a system
?SSID	Byte pointer to a 16 byte buffer to receive system ID
?SSBD	Byte pointer to a 16 byte buffer to receive the name of the disk the system was booted from

Figure 12.19 System information packet

Packet word ?SSID points to a buffer to which ?SINFO returns the name of the script file you specified during the system generation dialogue. Note that even though you may have renamed that file, ?SINFO always returns the *original* file name you used.

If either one of the fields ?SSID or ?SSBD is 0, no information is returned for that field.

Inputs

AC	Contents
AC2	Address of packet

Outputs

None

Options

None

Errors

Mnemonic	Meaning
ERIPT	Invalid packet type
ERMPR	Invalid packet address

Set the Searchlist *(library routine)*

?SLIST

Sets the searchlist to the specified list of fully-qualified pathnames.

The searchlist may contain up to five pathnames, which must be separated by commas, with no intervening blanks, and the list must be terminated by a null byte. (This is the format returned by the ?GLIST system call.)

Inputs

AC	Contents
AC0	Byte pointer to list of pathnames

Outputs

None

Errors

Mnemonic	Meaning
ERDOL	Device off line
ERFIL	Device read error
ERFTL	Filename too long
ERIFC	Invalid character in filename
ERIFT	Filename is not a directory
ERNAD	Non-directory name in pathname
ERPWL	Device write error
ERSTL	Searchlist is too long

?SPOS Set the Current File Position

Sets the file pointer for the specified I/O channel to a specific byte.

Normally, if you try to position past the current end of the file, the system will extend the file as needed. However, an attempt to exceed the file size will produce an error if

- You have specified the EF option (described below).
- You have opened a disk device as a file, since the end of file for a disk is a physical limitation. (There is no space left.)

Inputs

AC	Contents
AC0	Channel number
AC1	High order 16 bits of file position
AC2	Low order 16 bits of file position

Outputs

None

Options

Mnemonic	Meaning
EF	If the program attempts to position past the end of file, give an error return with code EREOF.

Errors

Mnemonic	Meaning
ERDOL	Device off line
EREOF	End of file encountered
ERFIL	Device read error
ERICN	Invalid channel number
ERIOD	Invalid operation for device
ERPWL	Device write error

Set File Attributes

?STATR

Sets the attributes of the specified file.

The filename must be terminated by a null byte. The attributes are defined in Table 12.10. The right half of the attribute word (bits 8-15) is used or reserved by the system. The left half (bits 0-7) is reserved for user-defined attributes.

The CH option causes the attributes to be set for the file open on the channel specified in AC0.

Mnemonic	Meaning
?ATPM	Permanent: the file may not be deleted or renamed while this bit is set to 1; set by the system for directories and root directories of disks.
?ATRD	Read protect: this file may not be read.
?ATWR	Write protect: this file may not be written; set by the system for directories and root directories of disks.
?ATAT	Attribute protect: the attributes of this file may not be changed. Set by the system for devices and root directories of disks only.

Table 12.10 File attributes

Inputs

AC	Contents
AC0	Byte pointer to pathname
AC1	Attribute word

Outputs

None

Options

Mnemonic	Meaning
CH	Input a channel number instead of a byte pointer to a pathname in AC0

Errors

Mnemonic	Meaning
ERATP	File is attribute protected
ERBTL	Buffer extends into system space
ERFDE	File does not exist
ERFTL	Filename too long
ERIAT	Invalid attribute word
ERIFC	Invalid character in filename
ERIRB	Buffer too short
ERNAD	Non-directory name in pathname

Set the Current System Time and Date

?STIME

Sets the system time and date to the specified value. You must use the MP/AOS internal format, a 32-bit number representing the number of seconds since midnight, January 1, 1900. (Use library routines ?FTOD and ?FDAY to convert the time and day, respectively, from conventional to MP/AOS internal format.)

Inputs

AC	Contents
AC0	High order 16 bits of system time
AC1	Low order 16 bits of system time

Outputs

None

Options

None

Errors

None

?STMP Set Up Data Channel Map

This call allows the user control over any of the data channel maps.

The input to AC0 is the logical data channel slot allocated to the user by ?ALMP. The input to AC1 is the physical page number corresponding to a user logical address as returned by ?GMRP.

?STMP sets up one data channel map slot at a time.

?STMP can be issued from an interrupt handler routine. Note that interrupts are always enabled upon completion of ?STMP.

Inputs

AC	Contents
AC0	User's logical data channel slot . The numbering scheme is identical to that used in the ?ALMP call requesting the allocation: 0-31 Data Channel Map A 32-63 Data Channel Map B 64-95 Data Channel Map C 96-127 Data Channel Map D 128-1151 BMC map
AC1	Physical page number (returned by ?GMRP)

Outputs

None

Options

None

Errors

None

Set the System Clock *(library routine)*

?STOD

Allows a program to adjust the system clock at runtime. ?STOD accepts a time of day expressed in seconds, minutes, hours, and sets the system clock to the values specified.

Inputs

AC	Contents
AC0	Seconds (range 0-59_{10})
AC1	Minutes (range 0-59_{10})
AC2	Hour (range 0-23 (midnight to 11 pm), expressed in octal)

Outputs

None

Options

None

Errors

Mnemonic	Meaning
ERANG	Range error

?TMSG Translate a CLI-Format Message *(library routine)*

Retrieves selected portions of an interprogram message in CLI format.

The specified message must be terminated by a null byte. This call uses a packet; its format is given in Figure 12.20 below.

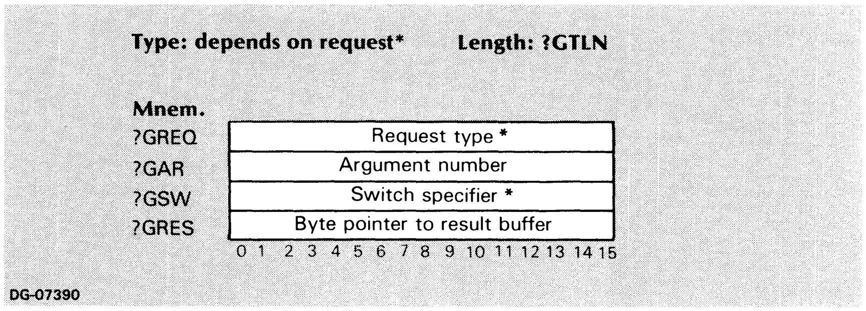


Figure 12.20 ?TMSG packet

**See Table 12.11 for request type.*

Inputs

AC	Contents
AC0	Byte pointer to message buffer
AC2	Address of packet

Outputs

AC	Contents
AC0	Depends on request type
AC1	Depends on request type

Options

None

Errors

Mnemonic	Meaning
ERBTL	Buffer extends into system space
ERIRB	Buffer too short
ERNAR	No argument for specified ?GAR.
ERNSS	No such switch (for ?GSWI only)

		Outputs	
Mnem.	**Meaning**	**AC0**	**AC1**
?GARG	Copy the argument specified by ?GAR to the result buffer.	Argument length	Unused
?GCMD	Get the message: issue a ?GTMSG call and place the message in the buffer pointed to by ?GRES.	Length of message	Unused
?GCNT	Get the argument count.	Number of arguments	Unused
?GSWC	Return the number of switches present on the specified argument.	Number of switches	
?GSWI	Test if the switch specified by the switch number in ?GSW is attached to the argument specified by ?GAR. If found, copy the switch value (if any) to the result buffer.	0 if the switch has no value. >0 for the length of the switch value.	Length of returned string.
?GSWM	Functions like ?GTSW, except that it tests for the presence of switches for which a minimum number of unique characters has been specified in AC1. E.g., if you use a /OPTIONS switch and you specify AC1=3, ?GSWM detects /OPT, /OPTI ... /OPTIONS, but not /OP.	Test result*	
?GSWS	Get the switch set: check for single-letter switches, and set the corresponding bits in AC0 and AC1.	Flags for /A through /P (bit 0 = /A, bit 1 = /B, bit 15 = /P)	Flags for /Q through /Z (bit 0 = /Q, bit 9 = /Z, bits 10-15 unused).
?GTSW	Test if the switch specified by the byte pointer in ?GSW is attached to argument specified by ?GAR. If so, copy its value (if any) to the result buffer.	Test result*	Unused

Table 12.11 Types of requests

**Test results are:*

-1 if the switch was not found.

0 if the switch has no value.

< 0 for the length of the switch value.

NOTES: *The command or program name is referenced as the 0th argument.*

?TMSG regards upper- and lower-case letters as equivalent on input; on output it converts all letters to uppercase.

?TPORT Port Translation

Allows a process to get the PID of another process with which it is communicating.

?TPORT successfully returns the other process' PID, even if that process has previously broken off the connection.

Inputs

AC	Contents
AC0	Port number used to communicate with the destination process

Outputs

AC	Contents
AC2	PID of process using this port

Options

None

Errors

Mnemonic	Meaning
ERPOR	Invalid port

?UNBLOCK

Unblock a Process

?UNBLOCK allows the process specified by AC2 to be rescheduled. This call will have no effect if the process was not previously blocked.

Inputs

AC	Contents
AC2	Process ID of process to be unblocked

Outputs

None

Options

None

Errors

Mnemonic	Meaning
ERPID	Invalid process ID

?UNPEND Resume Execution of a Task

Resumes execution of the specified task.

You can specify the task to be unpended either by its identifier or by a 16-bit event number. Event numbers must be greater than or equal to ?EVMIN and less than or equal to ?EVMAX.

If you specify an event number that several tasks are waiting for, only one task is unpended, unless you use the BD option to unpend all waiting tasks.

The system unpends tasks on event on a first in, first out basis: the first task pended is the first unpended, regardless of time remaining in its timeout interval.

You can also specify that the unpended task take the error return from its ?PEND or ?IPEND call.

Inputs

AC	Contents
AC0	Message word to unpended task(s)
AC2	Event code or task identifier (ID option)

Outputs

AC	Contents
AC0	Number of tasks unpended

Options

Mnemonic	Meaning
BD	Unpends all tasks waiting for this event
ER	Unpends task(s) at the error return
ID	AC2 contains task identifier, not event code

NOTE: *Do not specify the BD and ID options together.*

Errors

Mnemonic	Meaning
EREVT	Invalid event number
ERTID	Invalid task identifier

Write Data to a Device or File

?WRITE

Writes data to the device or file on the specified I/O channel. You can write data using either *dynamic* or *data-sensitive* mode. In *extended I/O,* data is written from a memory segment to a disk device.

To use dynamic writing, you must specify the number of bytes to be written.

If you are writing to a disk, you can improve the efficiency of your program by transferring entire disk blocks. To do this you must

- Set the file pointer to a multiple of 512 before the transfer.
- Specify a multiple of 512 bytes to write.
- Specify a buffer which is word aligned in your address space.

Data-sensitive writing is selected by the DS option. In this case, you specify the maximum number of bytes to transfer; the system then writes until it has either written one of the default *delimiters* — New-Line (12_8), Carriage Return (15_8), Form-Feed (14_8) or null (00_8) — or until it has written a delimiter specified by a *delimiter table.* (See ?SCHS for a discussion of delimiter tables.)

If you attempt to write past the end of file, your program will take the error return if you specified the EF option. If you did not specify the option, the system will extend the file as needed.

After the transfer, AC2 will contain the number of bytes written, whether or not the error return was taken.

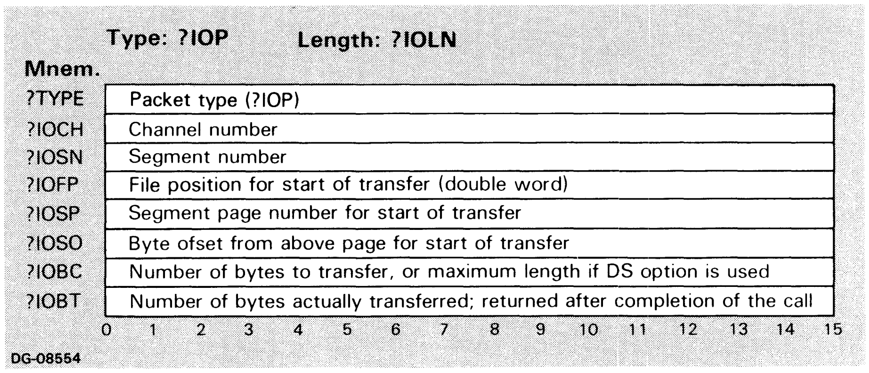
Extended I/O is selected by the PKT option. In this case you specify the address of a packet whose format is described in Figure 12.21.

This form of I/O transfers the data to or from a specified memory segment. The memory segment need not be mapped to the user's address space. To determine the segment address for the start of the transfer, the packet specifies a page number, as well as a byte offset.

Extended I/O may be either dynamic or data sensitive. It is *not* allowed on character devices. After the end of the transfer, two values are returned: word ?IOBT of the extended I/O packet returns the number of bytes transferred; double word ?IOFP is updated to reflect the current position in the file.

NOTE: *The block transfer optimization also applies to extended (PKT option) writes.*

The FL option delays the return of ?WRITE until all outstanding blocks associated with the channel have been flushed to disk. This option can be of use in establishing check points.



Mnem.	
?TYPE	Packet type (?IOP)
?IOCH	Channel number
?IOSN	Segment number
?IOFP	File position for start of transfer (double word)
?IOSP	Segment page number for start of transfer
?IOSO	Byte ofset from above page for start of transfer
?IOBC	Number of bytes to transfer, or maximum length if DS option is used
?IOBT	Number of bytes actually transferred; returned after completion of the call

Figure 12.21 Extended I/O packet

Inputs

AC	Contents
AC0	Channel number
AC1	Byte pointer to data to write
AC2	Byte count (dynamic) Options: DS: maximum byte count PKT: packet address

Outputs

AC	Contents
AC2	Number of bytes written Option: NP: System task ID

Options

Mnemonic	Meaning
EF	Cause an error return if the program attempts to write past the end of file
FL	Return after written data has been flushed to disk
NP	Non-pended call; (system task ID returned in AC2)
DS	Data-sensitive write
PKT	Extended write (not valid on character devices)

Errors

Mnemonic	Meaning
ERBTL	Buffer too long
ERDOL	Device off line
EREOF	End of file encountered
ERFIL	Device read error
ERICN	Invalid channel number
ERIRB	Buffer too short
ERLTL	Too many bytes without a delimiter (DS option only)
ERNOT	No free task control blocks (NP option only)
ERPWL	Device write error
ERWAD	Write access denied

?WRMEM Write to Memory *(Debugger call)*

Writes data into the memory of the program being debugged, starting at the address specified in AC0. (Specific memory locations in the target program can be examined via the ?RDMEM system call.)

Before issuing ?WRMEM, the caller must set up a buffer containing the data to be written. The buffer address and its word length are specified in AC2 and AC1 respectively.

Inputs

AC	Contents
AC0	Address in target program
AC1	Length in words of data to write
AC2	Address of buffer from which to write

Outputs

None

Options

None

Errors

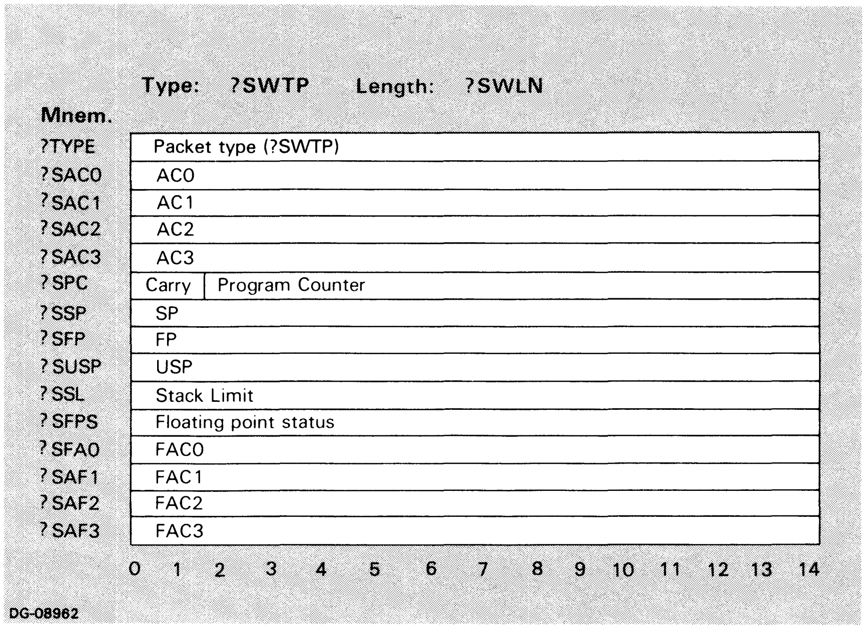
Mnemonic	Meaning
ERBTL	Buffer extends out of user space
ERNDB	No target program defined
ERTAD	Invalid target program address

Modify Task State *(Debugger call)*

?WRST

Permits a debugger program to modify part of the task status information for the task whose identifier is specified in AC0. Task identifers are retrieved via the ?GIDS system call; task status information for the specified task can be obtained via the ?RDST system call.

Before issuing ?WRST, the caller must set up a packet in the format described in Figure 12.22. The packet words specify modifiable task status values: contents of accumulators 0 through 3 (packet words ?SAC0 through ?SAC3), Program Counter and Carry bit values, (word ?SPC), the locations of the hardware stack and frame pointers (words ?SSP and ?SFP), and Unique Storage Position, and stack limit (words ?SUSP and ?SSL); packet words ?SFPS through ?SAF3 specify modifiable floating-point information, i.e., the floating point register status and contents of the floating point registers.



Type: ?SWTP Length: ?SWLN

Mnem.	
?TYPE	Packet type (?SWTP)
?SAC0	AC0
?SAC1	AC1
?SAC2	AC2
?SAC3	AC3
?SPC	Carry | Program Counter
?SSP	SP
?SFP	FP
?SUSP	USP
?SSL	Stack Limit
?SFPS	Floating point status
?SFA0	FAC0
?SAF1	FAC1
?SAF2	FAC2
?SAF3	FAC3

Figure 12.22 Write task state packet

Inputs

AC	Contents
AC0	Identifier of task whose state is to be modified
AC2	Address of packet

Outputs

None

Options

None

Errors

Mnemonic	Meaning
ERIPT	Invalid packet type
ERMPR	Invalid packet address
ERTID	Invalid task identifier

Intercept Signals and Resume Program Execution *(Debugger call)*

?WSIG

This call can only be issued by a debugger program, i.e., a program having previously invoked another program by means of the ?PROC system call with DB (debug) option.

AC0 sets the mask of signal classes to intercept. There are four signal classes, namely, user catchable signals, abort signals, breakpoints, and system calls. Tables 12.12 through 12.15 list the signal classes and the signals in each class.

Each signal and signal class is designated by a mnemonic corresponding to a number. The mnemonics and their numeric values are listed in OPARU.SR, the user parameter file. See Appendix E.

Class	**?SGUC**
?SGSO	Stack overflow
?SGFP	Floating exception
?SGCI	Commercial exception

Table 12.12 User catchable signals

NOTE: *When binding the program, the user can provide the address of an appropriate routine to handle each of these signals. When a user exception routine is entered, the user stack has a return block of the following format pushed on it:*

Stack return
block format

AC0
AC1
AC2
AC3
SP - stack pointer: to carry in bit 0, to program counter in bits 1 through 15

In the absence of user specified routines for these signals, MP/AOS force binds with the user program a routine to direct the signals to the system.

Class	?SGAS
?SGJ0	User JMP 0
?SGVT	Validity trap
?SGWP	Write protection trap
?SGIO	I/O protection trap
?SGIT	Indirection protection trap
?SGCB	^C^B console interrupt by user
?SGCE	^C^E console interrupt by user
?SGRI	Alpha reserved instruction trap
?SGRT	Program termination (?RETURN, all tasks ended)
?SGRB	Program termination (?RETURN BK, all tasks ended, checkpointed)
?SGKL	Program termination (?KILL, all tasks ended)
?SGKB	Program termination (?KILL BK, all tasks ended, checkpointed)

Table 12.13 Abort signals

Class	?SGSC
?SGCL	System call SVC*

Table 12.14 System calls

NOTE: *System calls ?IXIT and ?IUNPEND cannot be intercepted as signals by the debugger program.*

Class	?SGBP
?SGUS	Unknown SVC*
?SGBK	Breakpoint SVC**
?SGB2	Secondary
?SGCD	^C^D (Enter debugger)
?SGEX	?EXEC issued
?SGOL	Overlay loaded

Table 12.15 Breakpoint signals

**SYC, sometimes named SCL or SVC, is a hardware instruction that pushes a return block on the caller's user stack and transfers control to a kernel routine which activates either a system or breakpoint handling routine as appropriate. The SYC instruction is discussed in the Principles of Operation manual appropriate to your processor.*

***There are two breakpoint signals corresponding to two different SYC's:*

?SGBK	*Primary Breakpoints SYC 0,1*
?SGB2	*Secondary Breakpoints SYC 1,0*

When a signal is intercepted, the process being debugged is blocked, and control passes to the debugger program.

?WSIG returns the signal number of the intercepted signal, as well as the program counter and the task identifier of the task currently active in the program being debugged. The debugger program can obtain task identifiers (?GIDS), examine or modify task state (?RDST, ?WRST), or examine and modify memory locations (?RDMEM, ?WRMEM).

To resume program execution, another ?WSIG call must be issued. Issuing ?WSIG with the Resume (RE) option, causes the signal last trapped to be ignored and program execution to resume normally until the next signal is detected. This option affects only system calls and permits them to be executed rather than being trapped again upon program resumption.

NOTE: *In the case of the ?EXEC issued (?SGEX) and Overlay loaded (?SGOL) signals in the Breakpoints (?SGBP) class, the event is handled by the system before the signal comes in and the RE option has no effect.*

Inputs

AC	Contents
AC0	Mask of signal classes to intercept

Outputs

AC	Contents
AC0	Signal number
AC1	Program counter
AC2	Task identifier of currently active task

Options

Mnemonic	Meaning
RE	Resume system call

Errors

Mnemonic	Meaning
ERNSG	No outstanding signal

The ASCII Character Set

A

DECIMAL	OCTAL	HEX	KEY SYMBOL	MNEMONIC
0	000	00	↑@	NUL
1	001	01	↑A	SOH
2	002	02	↑B	STX
3	003	03	↑C	ETX
4	004	04	↑D	EOT
5	005	05	↑E	ENQ
6	006	06	↑F	ACK
7	007	07	↑G	BEL
8	010	08	↑H	BS (BACKSPACE)
9	011	09	↑I	TAB
10	012	0A	↑J	NEW LINE
11	013	0B	↑K	VT (VERT. TAB)
12	014	0C	↑L	FORM FEED
13	015	0D	↑M	CARRIAGE RETURN
14	016	0E	↑N	SO
15	017	0F	↑O	SI
16	020	10	↑P	DLE
17	021	11	↑Q	DC1
18	022	12	↑R	DC2
19	023	13	↑S	DC3
20	024	14	↑T	DC4
21	025	15	↑U	NAK
22	026	16	↑V	SYN
23	027	17	↑W	ETB
24	030	18	↑X	CAN
25	031	19	↑Y	EM
26	032	1A	↑Z	SUB
27	033	1B	ESC	ESCAPE
28	034	1C	↑\	FS
29	035	1D	↑]	GS
30	036	1E	↑↑	RS
31	037	1F	↑—	US

DG-05495

DECIMAL	OCTAL	HEX	KEY SYMBOL
32	040	20	SPACE
33	041	21	!
34	042	22	" (QUOTE)
35	043	23	#
36	044	24	$
37	045	25	%
38	046	26	&
39	047	27	' (APOS)
40	050	28	(
41	051	29	)
42	052	2A	*
43	053	2B	+
44	054	2C	, (COMMA)
45	055	2D	-
46	056	2E	. (PERIOD)
47	057	2F	/
48	060	30	0
49	061	31	1
50	062	32	2
51	063	33	3
52	064	34	4
53	065	35	5
54	066	36	6
55	067	37	7
56	070	38	8
57	071	39	9
58	072	3A	:
59	073	3B	;
60	074	3C	<
61	075	3D	=
62	076	3E	>
63	077	3F	?
64	100	40	@

DECIMAL	OCTAL	HEX	KEY SYMBOL
65	101	41	A
66	102	42	B
67	103	43	C
68	104	44	D
69	105	45	E
70	106	46	F
71	107	47	G
72	110	48	H
73	111	49	I
74	112	4A	J
75	113	4B	K
76	114	4C	L
77	115	4D	M
78	116	4E	N
79	117	4F	O
80	120	50	P
81	121	51	Q
82	122	52	R
83	123	53	S
84	124	54	T
85	125	55	U
86	126	56	V
87	127	57	W
88	130	58	X
89	131	59	Y
90	132	5A	Z
91	133	5B	[
92	134	5C	\
93	135	5D	]
94	136	5E	↑ OR ∧
95	137	5F	← OR —
96	140	60	` (GRAVE)

DECIMAL	OCTAL	HEX	KEY SYMBOL
97	141	61	a
98	142	62	b
99	143	63	c
100	144	64	d
101	145	65	e
102	146	66	f
103	147	67	g
104	150	68	h
105	151	69	i
106	152	6A	j
107	153	6B	k
108	154	6C	l
109	155	6D	m
110	156	6E	n
111	157	6F	o
112	160	70	p
113	161	71	q
114	162	72	r
115	163	73	s
116	164	74	t
117	165	75	u
118	166	76	v
119	167	77	w
120	170	78	x
121	171	79	y
122	172	7A	z
123	173	7B	{
124	174	7C	|
125	175	7D	}
126	176	7E	~ (TILDE)
127	177	7F	DEL (RUBOUT)

Figure A.1

DGC Standard Floating Point Format

B

Word for word, floating point format (Figure B.1) provides a much larger range than integer format, at the expense of some precision. It also provides the ability to operate on fractions. The maximum range of floating point format is equivalent to a 16-word multiple-precision integer. In addition, floating point operations are executed faster than most multiple-precision integer operations.

We represent a floating point value using a four-byte number for single precision or an eight-byte number for double precision. The four- or eight-byte aggregate contains three fields:

- a fractional part called the *mantissa* which is *normalized* at the end of all floating point operations; that is, the mantissa's value is adjusted to be greater than or equal to 1/16 and less than 1;
- an *exponent,* which is adjusted to maintain the correct value of the number;
- a *sign.*

To operate on a number in memory employing the floating point arithmetic instructions, the number must be *word aligned;* that is, bit 0 of the first byte of the number is bit 0 of the first word of a two-word or four-word area in memory.

The magnitude of a floating point number is defined as

MANTISSA X $16^{(\text{TRUE VALUE OF THE EXPONENT})}$

The magnitude of a single- or double-precision number is thus in the approximate range:

-7.237×10^{75} TO 7.237×10^{75}

We represent zero in floating point by a number with all bits zero, known as *true zero*. When a calculation results in a zero mantissa, the number is automatically converted to a true zero.

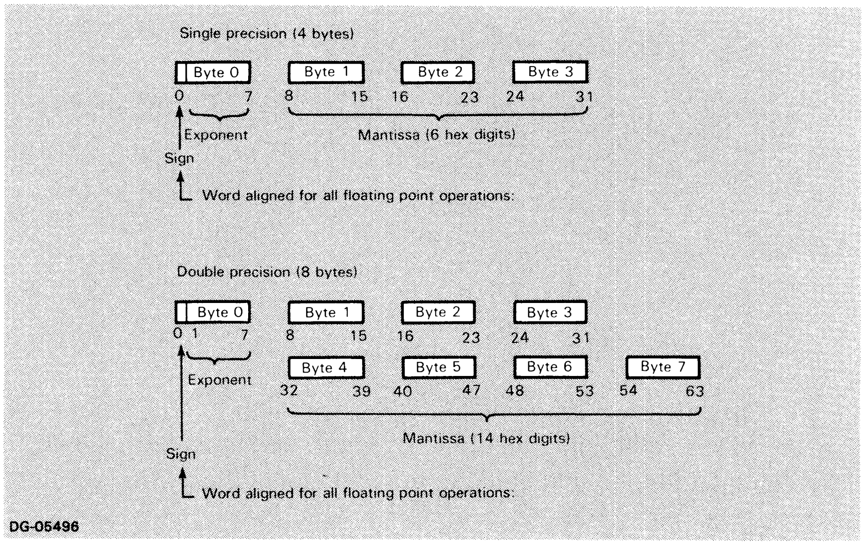


Figure B.1 Floating point formats

Sign

Bit 0 of the first byte is the sign bit. If the sign bit is 0, the number is positive. If the sign bit is 1, the number is negative.

Exponent

The right-most seven bits of the first byte contain the exponent. We use *excess 64* representation. For both positive and negative exponents, the value is the true value of the exponent plus 64. Table B.1 illustrates this.

Exponent Field	True Value of Exponent
0	-64
64	0
127	63

Table B.1 Excess 64 representation of exponents

Mantissa

Bytes 1 to 3 (single precision) or bytes 1 to 7 (double precision) contain the mantissa. By definition, the binary point lies *between* byte 0 and byte 1 of a floating point number. To keep the mantissa's value in the range 1/16 to 1, the results of each floating point calculation are *normalized*. A mantissa is normalized by shifting it left one hex digit (four bits) at a time, until the high-order four bits (the left-most four bits of byte 1) represent a nonzero quantity. For every hex digit shifted, the exponent decreases by one.

CLI Message Format

C

This appendix describes the format of messages passed to programs by the MP/AOS operating system. The syntax the CLI (Command Line Interpreter) provides to the user is more complex than that described here: a number of features, such as command repetition and filename templates, are interpreted by the CLI and not passed to programs. This appendix only describes the format of CLI messages as the program sees them. You can use the ?TMSG library routine to translate these messages.

For more information on the CLI command language, see *MP/AOS CLI Manual* (DGC No. 069-400201).

Arguments

CLI messages consist of a program name alone followed by one or more arguments. An argument may consist of a filename, function name or any other string of characters. If the message contains multiple arguments, they are separated by commas. The last argument (or the program name itself if there are no arguments) is always followed by a null byte.

Switches

Switches are modifiers that can follow the program name or any argument. All switches are preceded by a slash (/) and can consist of one or more characters. Switches have two forms: *simple* and *keyword*. A simple switch can take this form:

PROGRAM_NAME */switch_name*

A keyword switch can take this form:

PROGRAM_NAME */switch_name=value*

Value may be any number, filename, etc.

Consider the following example of a command to MASM, the MP/AOS Macroassembler program. You might type a CLI command on your console:

```
) X MASM/U/L=@LPT DEFS/S PROG
```

The X is the minimum unique abbreviation of XEQ. MASM is the name of the program you want to run. /U is a simple switch: it instructs MASM to include user symbols in the object file it generates. /L=@LPT is a keyword switch: it instructs MASM to send the listing file to the line printer (device name @LPT). The arguments DEFS and PROG are the names of two files to be assembled. The /S following DEFS is a simple switch: it tells MASM to skip that file on the second assembly pass.

As a result of this command the CLI creates an interprogram message in the form:

```
) MASM/U/L=@LPT,DEFS/S,PROG (plus a terminating null byte)
```

As you can see, the X has been removed. Also, all strings of spaces have been converted to single commas, and all lowercase characters have been converted to uppercase. A trailing null byte has been appended to terminate the message and tabs have been treated as blanks.

I/O Device Codes

D

The first two tables in this appendix contain device code assignments for ECLIPSE and microNOVA disk(ette) and magnetic tape devices supported under MP/AOS. Tables D.3 and D.4 list all standard ECLIPSE and microNOVA device codes.

Octal Device Codes	Mnemonic	Priority Mask Bit	Device Name	Model No.
17	LPT	12	PIO (programmed I/O) line printer	4034A/D,G/H or LP2 with PIO controller
17	LP2	12	Data channel (or PIO) line printer	LP2 with data channel controller
17	LPB	12	Data channel line printer	4215/19, 4244/45
22	MTA	10	Magnetic tape controller	6021/25, 6026A, 6027, 6125
26	DPH	7	12.5 or 25-Mbyte fixed disk and/or 1.25-Mbyte diskettes	6101/2/4/5 6097
27	DPF	7	50/95/190-Mbyte cartridge disk	6060/67
33	DPD	7	315-Kbyte diskette subsystem and/or 10-Mbyte cartridge disk	6030 6045/50/51
57	LPT1	12	Second PIO line printer	
57	LP21	12	Second data channel line printer	
57	LPB1	12	Second data channel line printer	
67	DPG	7	20-Mbyte cartridge disk	6070
73	DPD1	7	Second 315-Kbyte diskette subsystem and/or 10-Mbyte disk	

Table D.1 Device code assignments for ECLIPSE disk, diskette, line printer, and magnetic tape devices suppported by MP/AOS

Octal Device Code	Mnemonic	Priority Mask Bit	Device Name	Model No.
17	LPT	12	PIO (programmed I/O) line printer	4034A-D,G/H, or LP2 with PIO controller
17	LP2	12	Data channel (or PIO) line printer	LP2 with data channel controller
17	LPB	12	Data channel line printer	4215/19, 4244/5
22	MTA	10	Magnetic tape controller	6123
26	DPH	7	12.5 or 25-Mbyte fixed disk and/or 1.25-Mbyte diskettes	6098/99/6100/6103 6097
27	DPD	7	10-Mbyte cartridge disk	6095
33	DPX	10	315-Kbyte diskette subsystem	6038/6039
57	LPT1	12	Second PIO line printer	
57	LP21		Second data channel line printer	
57	LPB1		Second data channel line printer	
66	DPH1	7	Second 12.5 or 25-Mbyte fixed disk and/or 1.25-Mbyte diskettes	
67	DPD1	7	Second 10-Mbyte cartridge disk	
73	DPX1	10	Second 315-Kbyte diskette subsystem	

Table D.2 MicroNOVA device code assignments for disks, diskettes, line printers, and magnetic tape devices supported by MP/AOS

Octal Device Code	Mnem	Priority Mask Bit	Device Name
00	—	—	Unused
01	WCS	—	Writeable control store option [or APL register]
	[APL]	—	Error checking and correction
02	ERCC	—	Memory allocation and protection
	MAP		
03			
04			
05	BMC	—	Burst multiplexor channel
06	MCAT	12	Multiprocessor adapter transmitter
			Multiprocessor adapter receiver
07	MCAR	12	TTY input
			TTY output
10	TTI	14	
11	TTO	15	
12	PTR	11	Paper tape reader
13	PTP	13	Paper tape punch
14	RTC	13	Real-time clock
15	PLT	12	Incremental plotter
16	CDR	10	Card reader
17	LPT	12	Line printer
20	DSK	9	Fixed-head disc
21	ADCV	8	A-D converter
22	MTA	10	Magnetic tape
23	DACV	None	D-A converter
24	DCM	0	Data communications multiplexor
25			Fixed-head DG/Disc
26	DKB	9	DG/Disk storage subsystem
27	DPF	7	Asynchronous hardware multiplexor
30	QTY	14	
30	SLA	14	Synchronous line adapter
31[1]	IBM1	13	IBM 360/370 interface
32	IBM2	13	IBM 360/370 interface
33	DKP	7	Moving head disk
34	CAS[1]	10	Cassette tape
	DCU[4]	4	Data control unit
34	MX1	11	Multiline asynchronous controller
			Multiline asynchronous controller
35	MX2	11	Interprocessor bus—half-duplex
			IPB watchdog timer
36	IPB	6	IPB full-duplex input
			Synchronous communication receiver
37	IVT	6	
40[2]	DPI	8	
40	SCR	8	

Table D.3 Standard ECLIPSE I/O device codes

Octal Device Code	Mnem	Priority Mask Bit	Device Name
41[3]	DPO	8	IPB full-duplex output
41	SCT	8	Synchronous communication transmitter
			Digital I/O
42	DIO	7	Digital I/O timer
43	DIOT	6	Programmable interval timer
43	PIT	11	
44	MXM	12	Modem control for MX1/MX2
45			Second multiprocessor transmitter
46	MCAT1	12	Second multiprocessor receiver
			Second TTY input
47	MCAR1	12	
50	TTI1	14	
51	TTO1	15	Second TTY output
52	PTR1	11	Second paper tape reader
53	PTP1	13	Second paper tape punch
54	RTC1	13	Second real-time clock
55	PLT1	12	Second incremental plotter
56	CDR1	10	Second card reader
57	LPT1	12	Second line printer
60	DSK1	9	Second fixed-head disk
61	ADCV1	8	Second A-D converter
62	MTA1	10	Second magnetic tape
63	DACV1	None	Second D-A converter
64			
65	IOP1	5[5]	Host to IOP interface
66	DKB1	9	Second fixed-head DG/Disk
67	DPF1	7	Second DG/Disk storage subsystem
70	QTY1	14	Second asynchronous hardware multiplexor
			Second synchronous line adapter
70	SLA1	14	Second IBM 360/370 interface
			Second IBM 360/370 interface
71[1]			Second moving head disk
		13	
72			
	DKP1	7	
73			
74	CAS1	10	Second cassette tape
74[1]		11	Second multiline asynchronous controller
75		11	Second multiline asynchronous controller
76	DPU	4	DCU to host interface
77	CPU	—	Central processor and console functions

Table D.3 Standard ECLIPSE I/O device codes (continued)

[1] *Code returned by INTA and used by VCT.*

[2] *Can be set up with any unused even device code equal to 40 or above.*

[3] *Can be set up with any unused odd device code equal to 41 or above.*

[4] *Can be set to any unused device code between 1 and 76.*

[5] *Micro interrupts are not maskable.*

Octal Device Code	Mnemonic	Priority Mask Bit	Device Name
00	—	—	Returned by power monitor in response to INTA
01	APL	—	Auto program load register
02	PAR	—	Parity checking
03	MAP	—	Memory allocation and protection
04			
05			
06			
07			
10	TTI	14	TTY input
11	TTO	15	TTY output
12	PTR	11	Paper tape reader
13			
14	RTC	13	Real-time clock
15			
16			
17	LPT	12	Line printer
20			
21	ADCV	8	A-D converter
22	MTA	10	Magnetic tape
23	DACV	8	D-A converter
24	NVM	10	I/O memory
25	NVM1	10	I/O memory
26	DPH [or NVM2]	7 [10]	12.5- or 25-Mbyte disk and 1.25-Mbyte diskettes [or I/O memory]
27		7 [10]	10-Mbyte cartridge disk [or I/O memory]
30	DPD [or		
31	NVM3]		
32			
33		10	315-Kbyte diskette subsystem
34		8 [9]	Sync/async controller [sync/async controller]
	DPX		
	MUX [ASLM]		
35	CRC	8	Cyclic redundancy checker
36			
37			
40			

Table D.4 Standard microNOVA I/O device codes

Octal Device Code	Mnemonic	Priority Mask Bit	Device Name
41			
42	DIO	5	Digital I/O interface
43	PIT	11	Programmable interval timer
44	MUX1	8	Second sync/async controller
45	CRC	8	Second cyclic redundancy checker controller
46			
47	VID	6	Video interface
50	TTI1	14	Async controller/remote restart receiver
51	TTO1	15	Async controller/remote restart transmitter
52	PTR1	11	Second paper tape reader
53			Second real-time clock
54	RTC1	13	
55			
56			
57	LPT1	12	Second line printer
60			
61	ADCV1	8	Second A-D interface
62	MTA1	10	Second magnetic tape
63	DACV1	8	Second D-A interface
64	SNVM	10	I/O memory
65	SNVM1	10	I/O memory
66	DPH1 [SNVM2]	7 [10]	Second 12.5- or 25-Mbyte disk and 1.25-Mbyte diskettes [I/O memory]
67		7 [10]	Second 10-Mbyte cartridge disk [I/O memory]
	DPD1 [SNVM3]		
70			
71			
72			
73	DPX	10	Second 315-Kbyte diskette subsystem
74	ASLM1	9	Second sync/async controller
75			
76			
77	CPU	—	Central processor and console functions

Table D.4 Standard microNOVA I/O device codes (continued)

User Parameter Files

E

All MP/AOS systems include a set of *parameter files.* These are assembler source files that contain symbol definitions without any executable code. You use these files to prepare permanent symbol tables for the Macroassembler, using the /S function switch which skips the second assembly pass (produces no .OB file) and saves the Macroassembler's symbol table, renaming it MASM.PS. For more detailed information on /S, see *MP/AOS Macroassembler, Binder, and Library Utilities* (DGC No. 069-400210).

Table E.1 describes the MP/AOS parameter files.

This appendix contains an assembled copy of three parameter files, OPARU.SR, MP_OS_ERCOD.SR, and MP_AOS_ERCOD.SR. (See Figures E.1 through E.3). Refer to the listing to find out the numeric value of an MP/AOS symbol. The list of error codes in MP_OS_ERCOD.SR and MP/AOS_ERCOD.SR is particularly useful when debugging programs.

NOTE: *This listing of OPARU.SR, MP_OS_ERCOD.SR, and MP/AOS_ERCOD.SR reflects the state of the system at the time this manual was printed. The values of some mnemonics may change from time to time. If there is any doubt about the value of a symbol, check the copies of OPARU.SR, MP_OS_ERCOD.SR, and MP/AOS_ERCOD.SR that were released with your system, as well as your latest release notice.*

File	Contents
EBID.SR	The basic ECLIPSE instruction set
MP/AOS_ERCOD.SR	Additional MP/AOS error codes
MP_OS_ERCOD.SR	MP/OS error codes
NSKID.SR	Conditional skip mnemonics that simplify programming and improve program readability
OPARU.SR	Mnemonics used with system calls
OSYSID.SR	Definitions of system call numbers
SCALL.SR	Definition of system calls

Table E.1 Parameter files

```
 0001 OPARU    MP/MASM Assembler    Rev 03.10  04/12/82 14:59:30
                 .title oparu
02
03              ; ===================================
04              ; OPARU - User Parameter file
05              ; ===================================
06
07              ;  Symbol definition requirements are the following:
08              ;  1) All symbols of the form ?xxxx are reserved for
09              ;     defining parameters used jointly by the operating
10              ;     system and user assembled programs.
11              ;  2) All symbols of the form ERxxx are reserved for
12              ;     defining operating system error codes.
13              ;  3) Don't add symbols to OPARU unless they conform to
14              ;     these rules.
15
16               .ejec

 0002 OPARU
01
02       000000 i=0
03       000000 .dusr FQ1 = i
04       000001 .dusr FQ2 = i
05       000002 .dusr FQ3 = i
06       000003 .dusr FQ4 = i
07
08
09
10       000100 .dusr ?NMCH = 64.        ; Number of channels
11
12       000000 .dusr ?INCH = 00         ; Default console input channel
13       000001 .dusr ?OUCH = 01         ; Default console output channel
```

```

000001 .dusr ?EVMIN = 01            ; Minimum user pend event
075777 .dusr ?EVMAX = 075777        ; Maximum user pend event
076000 .dusr ?EDMI = 076000         ; minimum DG event code
077777 .dusr ?EDMX = 077777         ; maximum DG event code
076000 .dusr ?EVCH = ?EDMI          ; Channel 0 ^C^A event code

000017 .dusr ?MXFL = 15.            ; Maximum filename length
000177 .dusr ?MXPL = 127.           ; Maximum pathname length
000017 .dusr ?SVNL = 017            ; maximum server name length
000003 .dusr ?MXLD = 03             ; maximum link depth
000077 .dusr ?MXLL = 63.            ; maximum link length
000005 .dusr ?MXSL = 5              ; maximum number of searchlist entries
000200 .dusr ?MXSP = 128.           ; Maximum segment size (in pages)
000400 .dusr ?mxtk = 256.           ; maximum number of tasks
000040 .dusr ?mxov = 32.            ; max number of user overlays
                                    ; per process
000010 .dusr ?mxsw = 8.             ; max swap level
000377 .dusr ?MXPR = 255.           ; lowest priority, highest value
                                    ; specifiable

       ; FILE TYPES

177772 .dusr ?DLPT = -6             ; Lineprinter
177773 .dusr ?DCHR = -5             ; Character device
177774 .dusr ?DDVC = -4             ; Disk (directory device)
177775 .dusr ?DDIR = -3             ; Directory
177776 .dusr ?DMSG = -2             ; message file
177777 .dusr ?DPSH = -1             ; swap (push) file

000000 .dusr ?DSMN = 00             ; System min
000100 .dusr ?DSMX = 0100           ; System max
000101 .dusr ?DUMN = ?DSMX+1        ; User minimum
000200 .dusr ?DUMX = 0200           ; User maximum

       ;      SYSTEM TYPES

000000 .dusr ?DOBF = ?DSMN          ; OB file
000001 .dusr ?DSTF = ?DOBF+1        ; Symbol table file
000002 .dusr ?DPRG = ?DSTF+1        ; Program file
000003 .dusr ?DOLF = ?DPRG+1        ; Overlay file
000004 .dusr ?DBPG = ?DOLF+1        ; Bootable program file
000005 .dusr ?DPST = ?DBPG+1        ; Permanent symbol file (x.PS)
000006 .dusr ?DLIB = ?DPST+1        ; Library file (x.LB)
000007 .dusr ?DUDF = ?DLIB+1        ; User data file
```

```
000010 .dusr ?DTXT = ?DUDF+1         ; Text File
000011 .dusr ?DBRK = ?DTXT+1         ; Break file

0003 OPARU
000012 .dusr ?DIDF = ?DBRK+1         ; MP/ISAM data file
000013 .dusr ?DIXF = ?DIDF+1         ; MP/ISAM index file
000014 .dusr ?DLNK = ?DIXF+1         ; Link
000015 .dusr ?DBBS = ?DLNK+1         ; MP/Business Basic save file
000016 .dusr ?DMBS = ?DBBS+1         ; MP/Basic save file
000100 .dusr ?DLOG = ?DSMX           ; System log file



       ; MP/OS FILE ATTRIBUTES

000001 .dusr ?ATPM = 01     ; Permanent file, can't be deleted
000002 .dusr ?ATRD = 02     ; Can't be  read
000004 .dusr ?ATWR = 04     ; Can't be written
000010 .dusr ?ATAT = 8.     ; Attributes can't be changed
000020 .dusr ?ATDC = 16.    ; Delete on last close (user can't set)
000100 .dusr ?ATZR = 64.    ; Don't zero blocks on allocation



       ; MP/OS DEVICE CHARACTERISTICS

000000 i=0

100000 .dusr ?CST = 1B0   ; Simulate tabs if asserted
040000 .dusr ?CNAS = 1B1  ; If asserted, device is not ANSI standard
020000 .dusr ?CESC = 1B2  ; If asserted, handle escape as ^C^A sequence
010000 .dusr ?CECH = 1B3  ; If asserted, echo input to output
004000 .dusr ?CLST = 1B4  ; If asserted, echo form feed as ^L
002000 .dusr ?CBIN = 1B5  ; If asserted, input is in binary form (8 bit)
001000 .dusr ?C605 = 1B6  ; If asserted, device is 605x series
000400 .dusr ?CUCO = 1B7  ; Convert lowercase as uppercase
000200 .dusr ?C8BT = 1B8  ; 8 bit characters
000100 .dusr ?CNED = 1B9  ; Do not echo delimiters
000040 .dusr ?CEMM = 1B10 ; Echo characters exactly as input
000020 .dusr ?CICC = 1B11 ; Ignore control characters (0except
                          ;     delimiters and system chars)


       ;
       ; MP/OS USER DEFINABLE CHANNEL CHARACTERISTICS.
       ;

000000 i=0
```

```

177700 .dusr DUMO = 1777B9           ; Place holder
000040 .dusr ?CUCI = 1B10            ; convert input to uppercase

       ;
       ; MP/OS CHARACTER DEVICE HARDWARE CHARACTERISTICS WORD.
       ;

000001 .dusr ?BSTP = 1      ; placement of stop bit description
000003 .dusr ?BPAR = 3      ; placement of parity description
000005 .dusr ?BLVL = 5      ; placement of code level
000017 .dusr ?BRAT = 15.    ; placement of code rate

000000 i=0

140000 .dusr ?CSTP = 3B1             ; Stop bit mask
030000 .dusr ?CPAR = 3B3             ; Parity mask
006000 .dusr ?CLVL = 3B5             ; Code level mask
001760 .dusr RES1 = 77B11            ; reserved

0004 OPARU
000017 .dusr ?CRAT = 17B15           ; Line rate mask

040000 .dusr ?C1S = 040000           ; 1 Stop bit
140000 .dusr ?C2S = 0140000          ; 2 Stop bits
100000 .dusr ?C15S = 100000          ; 1.5 Stop bits

000000 .dusr ?CNPR = 00              ; No parity
010000 .dusr ?CODD = 010000          ; Odd parity
030000 .dusr ?CEVN = 030000          ; Even parity

000000 .dusr ?C5BC = 00              ; 5 bits
002000 .dusr ?C6BC = 02000           ; 6 bits
004000 .dusr ?C7BC = 04000           ; 7 bits
006000 .dusr ?C8BC = 06000           ; 8 bits

       ;
       ; ALM clock selection.
       ;
000011 .dusr ?cck0 = 9.              ; clock 0
000017 .dusr ?cck1 = 15.             ; clock 1
000000 .dusr ?cck2 = 0               ; clock 2
000001 .dusr ?cck3 = 1               ; clock 3
       ;
       ; Baud rate selection (not valid for ALMs).
```

```
              ;
       000000 .dusr ?C0050 = 00          ; Baud rate is 50
       000001 .dusr ?C0075 = 01          ; Baud rate is 75
       000002 .dusr ?C0110 = 02          ; Baud rate is 110
       000003 .dusr ?C1345 = 03          ; Baud rate is 134.5
       000004 .dusr ?C0150 = 04          ; Baud rate is 150
       000005 .dusr ?C0300 = 05          ; Baud rate is 300
       000006 .dusr ?C0600 = 06          ; Baud rate is 600
       000007 .dusr ?C1200 = 07          ; Baud rate is 1200
       000010 .dusr ?C1800 = 8.          ; Baud rate is 1800
       000011 .dusr ?C2000 = 9.          ; Baud rate is 2000
       000012 .dusr ?C2400 = 10.         ; Baud rate is 2400
       000013 .dusr ?C3600 = 11.         ; Baud rate is 3600
       000014 .dusr ?C4800 = 12.         ; Baud rate is 4800
       000015 .dusr ?C7200 = 13.         ; Baud rate is 7200
       000016 .dusr ?C9600 = 14.         ; Baud rate is 9600
       000017 .dusr ?C192K = 15.         ; Baud rate is 19200
              .dusr ?C1800 = 8.          ; Baud rate is 1800
              ;=======================================================
              ;
              ; Log file event codes
              ;
              ;===============================Baud rate is ==========

       000001 .dusr ?LSTR = 1            ;Logging start
       000002 .dusr ?LEND = 2            ;Logging end
       000004 .dusr ?LDER = 4            ;Device error
       001777 .dusr ?LXYZ = 1023.        ;Memory error

              ;========================================================
              ;
              ;    MP/OS packets are all typed.  The zero'th word of each and
              ;    every packet must contain the type code for that packet.
              ;    The actual packet begins at offset 1.  The packet length
              ;    includes the type word.
              ;

0005 OPARU
              ;========================================================
              ;
              ;      MP/OS packet types

       000000 .dusr ?PIP = 00      ; Rev 0 program information packet
       000001 .dusr ?TDP = 01      ; Rev 0 task definition packet
       000002 .dusr ?FSP = 02      ; Rev 0 file status packet
```

```
000003 .dusr ?DSP = 03      ; Rev 0 disk status packet
000004 .dusr ?ISP = 04      ; Rev 0 MP/ISAM call packet
000045 .dusr ?PRP = 045     ; Mp/aos proc packet
000046 .dusr ?EPIP = 046    ; Mp/aos extended info packet
000047 .dusr ?SIP = 047     ; Mp/aos system information packet
000410 .dusr ?SRTP = 0410   ; Mp/aos read task status packet
000411 .dusr ?SWTP = 0411   ; Mp/aos write task status packet
000412 .dusr ?SRP = 0412    ; Mp/aos IPC packet
000413 .dusr ?S_RP = 0413  ; Mp/aos send/receive packet
000414 .dusr ?ITYP = 0414   ; Mp/aos IDEF driver packet
000415 .dusr ?LTYP = 0415   ; Mp/aos line IDEF driver packet
000416 .dusr ?HSTP = 0416   ; Mp/aos histogramming packet
000417 .dusr ?HDTP = 0417   ; Mp/aos histogramming termination packet
000420 .dusr ?MSP = 0420    ; Mp/aos map segment
000421 .dusr ?IOP = 0421    ; Mp/aos segment I/O
000422 .dusr ?GSTP = 0422   ; Mp/aos get statistics packet



       ;==============================================
       ;
       ;      THE PROGRAM INFORMATION PACKET USED BY THE ?INFO CALL
       ;
       ;==============================================

000000 i=0

000000 .dusr ?TYPE = i           ;Offset of type word in the packet
000001 .dusr ?PPMN = i           ; Lowest pure (Ocode) address
000002 .dusr ?PPMX = i           ; Highest pure address
000003 .dusr ?PIMN = i           ; Lowest impure (Odata) address
000004 .dusr ?PIMX = i           ; Highest impure address
000005 .dusr ?PREV = i           ; Program revision number
000006 .dusr ?PLEV = i           ; Program level
000007 .dusr ?PHMA = i           ; Highest memory available
000010 .dusr ?POCH = i           ; Open channel mask
000011 .dusr ?PLN = 1?n4


       ;==============================================
       ;
       ;      THE TASK DEFINITION PACKET USED BY THE ?CTASK CALL
       ;
       ;==============================================

000000 i=0

000001 .dusr ?TPRI = i          ; Task priority (00 =< x =< 255)
000002 .dusr ?TSTA = i          ; Task starting address
000003 .dusr ?TSTB = i          ; Stack base
000004 .dusr ?TSTL = i          ; Stack limit
000005 .dusr ?TSTE = i          ; Stack error handler
                                ; (00=>system default)

0006 OPARU
000006 .dusr ?TAC2 = i          ; New task's initial ac2
000007 .dusr ?TUSP = i          ; New task's initial ?usp
000010 .dusr ?TKPP = i          ; Task kill post-processing
                                ;  (00 => none)

000011 .dusr ?TLN = 1?n5

       ;=================================================
       ;
       ;      THE FILE STATUS PACKET USED BY THE ?FSTAT CALL
       ;
       ;=================================================

000000 i=0

000001 .dusr ?FTYP = i          ; File type
000002 .dusr ?FATR = i          ; Attributes
000003 .dusr ?FESZ = i          ; Element size
000004 .dusr ?FTLA = i          ; Time last accessed (two words)
000006 .dusr ?FTLM = i          ; Time last modified (two words)
000010 .dusr ?FLEN = i          ; File length in bytes (two words)
000012 .dusr ?FLN = 1?n6


       ;=================================================
       ;
       ;      THE MESSAGE PACKET USED BY THE ?TMSG CALL
       ;
       ;=================================================

000000 i=0

000000 .dusr ?GREQ = i          ;Packet/request type  (see below)
000001 .dusr ?GNUM = i          ; Argument number
000002 .dusr ?GSW = i           ; Switch specifier
```

```
        000003 .dusr ?GRES = i          ; B.P. to buffer receiving switch
        000004 .dusr ?GTLN = 1?n7

               ;      PACKET / REQUEST TYPES  (?GREQ)

        000000 .dusr ?GCMD = 00         ; Get entire message
        000001 .dusr ?GCNT = ?GCMD+1    ; Get argument count
        000002 .dusr ?GARG = ?GCNT+1    ; Get argument
        000003 .dusr ?GTSW = ?GARG+1    ; Test a switch
        000004 .dusr ?GSWS = ?GTSW+1    ; Get (alphabetic) switches
        000005 .dusr ?GSWI = ?GSWS+1    ; Test for switch # ?GSW


               ;===================================================
               ;
               ;      The disk status packet used by the ?DSTAT call
               ;
               ;===================================================

        000000 i=0                   ; Disk status word (see below)
        100000 .dusr ?DWRP = 1B0     ; Disk is write protected
        040000 .dusr ?DLE1 = 1B1     ; Primary label block is bad
        020000 .dusr ?DLE2 = 1B2     ; Secondary label block is bad
        010000 .dusr ?DME1 = 1B3     ; Primary MDV block is bad
        004000 .dusr ?DME2 = 1B4     ; Secondary MDV block is bad

 0007 OPARU

        000000 i=0

        000001 .dusr ?DFB = i        ; Two word # of free blocks
        000003 .dusr ?DAB = i        ; Two word # of allocated blocks
        000005 .dusr ?DTMX = i       ; Maximum possible # of files
        000006 .dusr ?DTAL = i       ; Current # of allocated DITs
        000007 .dusr ?DSTW = i       ; Disk status word
        000010 .dusr ?DRER = i       ; Number of recoverable disk errors
        000011 .dusr ?DUER = i       ; Number of unrecoverable disk errors
        000012 .dusr ?DLN = 1?n9
```

```
       ;============================================
       ;
       ;             The packet used by the ?PROC call
       ;
       ;============================================


000000 i=0
000001 .dusr ?RMBP = i   ; Bytepointer to message
000002 .dusr ?RMLN = i   ; Length of message
000003 .dusr ?RCH0 = i   ; Bytepointer to channel 0
000004 .dusr ?RCH1 = i   ; Bytepointer to channel 1
000005 .dusr ?RSLI = i   ; Bytepointer to searchlist
000006 .dusr ?RDIR = i   ; Bytepointer to working dir.
000007 .dusr ?RPRI = i           ; Initial priority
000010 .dusr ?RMCH = i           ; Max. number of channels
000011 .dusr ?RMTC = i           ; Max. number of TCBs
000012 .dusr ?RMEM = i           ; Max. memory
000013 .dusr ?RMAS = i           ; Max number of attached segments
000014 .dusr ?RMON = i           ; Max number of overlay nodes
000015 .dusr ?PRLN = 1?n10


       ;============================================
       ;
       ;     Extended ?INFO packet
       ;
       ;============================================

000000 i=0                       ; Process status
100000 .dusr ?PBLK = 1B0         ; Process is blocked
077770 .dusr RES2 = 7777B12      ; reserved
000004 .dusr ?PROT = 1B13        ; Process is the root process
000001 .dusr ?efln = 1?n11

       ; ?PPMN - ?PHMA are the same as the ?INFO call packet

000000 i=0
000001 .dusr ?PTIM = i           ; Elapsed time
000003 .dusr ?PCPU = i           ; CPU time
000005 .dusr ?PBIO = i           ; I/O blocks
000007 .dusr ?PCIO = i           ; Characters transfered
000011 .dusr ?PCHN = i           ; Number of open channels
000012 .dusr ?PTSK = i           ; Number of active tasks

0008 OPARU
000013 .dusr ?PASG = i           ; Number of attached segments
```

```
000014 .dusr ?PSTS = i          ; Process status
000015 .dusr ?PPRI = i          ; Process priority
000016 .dusr ?ELN = 1?n12
       ;=========================================================
       ;
       ;      System information packet
       ;
       ;=========================================================

000000 i=0
000001 .dusr ?SREV = i     ; System rev. number
000002 .dusr ?SMEM = i     ; Number of memory pages (1k)
000003 .dusr ?SFMP = i     ; Number of free memory pages
000004 .dusr ?SPRC = i     ; Number of concurrent procs
000005 .dusr ?SIDS = i     ; Bytepointer to buffer for system ID
000006 .dusr ?SSBD = i     ; Bytepointer to buffer for system
000007 .dusr ?SLN = 1?n13



       ;=========================================================
       ;
       ;      Debugger signals and classes
       ;
       ;=========================================================

       ;  Breakpoint signals

020000 .dusr ?SGBP = 020000     ; Breakpoint signal class

000000 .dusr ?SGUS = 00         ; Unknown SVC
000001 .dusr ?SGBK = 01         ; Primary breakpoint SVC
000002 .dusr ?SGB2 = 02         ; Secondary breakpoint SVC
000003 .dusr ?SGCD = 03         ; ^C^D (enter debugger)
000004 .dusr ?SGEX = 04         ; ?EXEC issued
000005 .dusr ?SGOL = 05         ; Overlay loaded

       ; System call signals

040000 .dusr ?SGSC = 040000     ; System call signal class

000006 .dusr ?SGCL = 06         ; System call SVC

       ; ====================================================
       ;
       ; Signals 7 - 17 are used by MP/AOS,
       ; but a debugger never sees them.
       ;
       ; ==========================================

       ; User catchable signals

010000 .dusr ?SGUC = 010000     ; User catchable signal class

000020 .dusr ?SGSO = 020        ; Stack overflow
000021 .dusr ?SGFP = 021        ; Floating point exception
000022 .dusr ?SGCI = 022        ; Commercial instruction exception

       ; Abort signals

0009 OPARU

100000 .dusr ?SGAS = 0100000    ; Abort signal class

000023 .dusr ?SGJO = 023        ; User jump 0
000024 .dusr ?SGVT = 024        ; Validity trap
000025 .dusr ?SGWP = 025        ; Write protection trap
000026 .dusr ?SGIO = 026        ; I/O protection trap
000027 .dusr ?SGIT = 027        ; Indirection protection trap
000030 .dusr ?SGCB = 030        ; ^C^B
000031 .dusr ?SGCE = 031        ; ^C^E
000032 .dusr ?SGRI = 032        ; Reserved instruction trap
000033 .dusr ?SGRT = 033        ; Process termination ( ?RETURN )
000034 .dusr ?SGRB = 034        ; Process termination ( ?RETURN BK )
000035 .dusr ?SGKL = 035        ; Process termination ( ?KILL )
000036 .dusr ?SGKB = 036        ; Process termination ( ?KILL BK )


       ;============================================
       ;
       ;      Write task status packet
       ;
       ;============================================

       ; This packet is identical to the read task status packet for offsets
       ;  ?SACO thru ?SAF3


000000 i=0
000001 .dusr ?SACO = i          ; Tasks ACO
```

```
000002 .dusr ?SAC1 = i           ; Tasks AC1
000003 .dusr ?SAC2 = i           ; Tasks AC2
000004 .dusr ?SAC3 = i           ; Tasks AC3
000005 .dusr ?SPCC = i           ; PC and carry
000006 .dusr ?SSSP = i           ; Stack pointer
000007 .dusr ?SSFP = i           ; Frame pointer
000010 .dusr ?SUSP = i           ; USP
000011 .dusr ?SSSL = i           ; Stack limit
000012 .dusr ?SFPS = i           ; Floating point status
000014 .dusr ?SFA0 = i           ; Floating AC0
000020 .dusr ?SFA1 = i           ; Floating AC1
000024 .dusr ?SFA2 = i           ; Floating AC2
000030 .dusr ?SAF3 = i           ; Floating AC3

000034 .dusr ?SWLN = 1?n14  ; Length of write status packet


       ;=========================================================
       ;
       ;      Read task status packet
       ;
       ;=========================================================

000000 i=0               ; Tasks status
100000 .dusr ?STWT = 1B0         ;B13 Task is waiting
040000 .dusr ?STDR = 1B1         ;B14 - Task did a ?DRSCH
020000 .dusr ?STUS = 1B2         ;B15- Task is in user space

000000 i=0
000000 .dusr ?scmn = i     ; common fields in read and write status pkt
000034 .dusr ?stst = i     ; task status

0010 OPARU
000035 .dusr ?SPRI = i           ; Task priority
000036 .dusr ?SPNK = i           ; Tasks pend key
000037 .dusr ?STMO = i           ; Timeout
000041 .dusr ?STOL = i           ; Last overlay loaded
000042 .dusr ?SRLN = 1?n16
```

```
          ;==============================================
          ;
          ;      IPC packet format
          ;
          ;==============================================

000000 i=0

000001 .dusr ?ITLM = i          ; Timeout in seconds
000002 .dusr ?IMAD = i          ; Message byte address
000003 .dusr ?IMLN = i          ; Message length (bytes)
000004 .dusr ?IPCLN = 1?n17



          ;==============================================
          ;
          ;      Send/Receive packet
          ;
          ;==============================================

000000 i=0

000002 .dusr ?ISAD = i          ; Send message address
000003 .dusr ?ISDL = i          ; Send message length (Obytes)
000004 .dusr ?IRAD = i          ; Receive buffer address
000005 .dusr ?IRLN = i          ; Receive buffer length (Obytes)
000006 .dusr ?ISRLN = 1?n18


          ;==============================================
          ;
          ;      ?IDEF   offsets
          ;
          ;==============================================

000000 i=0
000001 .dusr ?IHND = i          ; handler address
000002 .dusr ?IMSK = i          ; Mask word
000003 .dusr ?ISTK = i          ; Stack address
000004 .dusr ?ISTL = i          ; Stack length
000005 .dusr ?IDAT = i          ; AC2 at int. time
000006 .dusr ?IHPR = i          ; Power recovery address
000007 .dusr ?ITLN = 1?n19
          ;==============================================
          ;
```

```
        ;     Line IDEF packet offsets
        ;
        ;==========================================

 0011 OPARU

000000 i=0
000001 .dusr ?LHND = i          ; Handler address
000002 .dusr ?LSTK = i          ; Stack address
000003 .dusr ?LSTL = i          ; Stack length
000004 .dusr ?LDAT = i          ; AC2 at int. time
000005 .dusr ?LHPR = i          ; Power recovery address
000006 .dusr ?LTLN = 1?n20


        ;==========================================
        ;
        ;     Histogram start packet
        ;
        ;==========================================

000000 i=0
000001 .dusr ?HTIK = i          ; Ticks per second
000002 .dusr ?HBUF = i          ; Buffer address
000003 .dusr ?HLEN = i          ; Buffer length
000004 .dusr ?HSAD = i          ; Starting hist. address
000005 .dusr ?HEAD = i          ; Ending hist. address
000006 .dusr ?HELN = 1?n21

        ;==========================================
        ;
        ;     Histogram stop packet
        ;
        ;==========================================

000000 i=0
000001 .dusr ?HETH = i          ; Elapsed time high
000002 .dusr ?HETL = i          ; Elapsed time low
000003 .dusr ?HSTH = i          ; Time in system high
000004 .dusr ?HSTL = i          ; Time in system low
000005 .dusr ?HUTH = i          ; Time in user high
000006 .dusr ?HUTL = i          ; Time in user low
000007 .dusr ?HIDH = i          ; Time in idle loop high
000010 .dusr ?HIDL = i          ; Time in idle loop low
000011 .dusr ?HDLN = 1?n22
```

```


          ;==============================================
          ;
          ;       Map segment packet
          ;
          ;==============================================

  000000 i=0
  000001 .dusr ?MSSN = i          ; Segment number
  000002 .dusr ?MSSP = i          ; Segment page number
  000003 .dusr ?MSPP = i          ; Process page number
  000004 .dusr ?MSNP = i          ; Number of pages

          ;==============================================
          ;
          ;       Segment I/O packet
          ;
          ;==============================================


0012 OPARU
  000000 i=0
  000001 .dusr ?IOCH = i          ; Channel number
  000002 .dusr ?IOSN = i          ; Segment number
  000003 .dusr ?IOFP = i          ; file position for start of transfer
                                  ; updated to reflect cur position
  000005 .dusr ?IOSP = i          ; Segment page number
  000006 .dusr ?IOSO = i          ; byte offset from above page number
  000007 .dusr ?IOBC = i          ; Number of bytes to transfer
  000010 .dusr ?IOBT = i          ; count of bytes actually transfered


          ;=====================================================
          ;
          ;       Get system statistics packet
          ;
          ;=====================================================


  000000 i=0

  000001 .dusr ?GIDT = i          ; Idle time (milliseconds)
  000003 .dusr ?GSYT = i          ; Time spent in system space

```

```



       ;======================================================
       ;
       ;    All of the following parameters/macros are assembler
       ;    parameters and do not need to be included for Pascal
       ;
       ;======================================================




000062 .dusr ?STKMIN = 50.      ; Minimum stack size
000016 .dusr ?USP = 016         ; User stack pointer


       ; Define the default frame pointer relative offsets into the save block

177774 .dusr ?OAC0 = -4         ; Original AC0
177775 .dusr ?OAC1 = -3         ; Original AC1
177776 .dusr ?OAC2 = -2         ; Original AC2
177777 .dusr ?OFP = -1          ; Original Frame Pointer
000000 .dusr ?ORTN = 00         ; Return address and Carry bit

000001 .dusr ?TMP = 01          ; First free stack loc rel to FP



       ;======================================================
       ;
       ;        the real-time program description block;
       ;         this is a packet produced for programs
       ;         bound with the /SA or /SP switches.  It
       ;         is based on the symbol ?ZSPA.
       ;         MP/AOS programs only.
       ;======================================================
```

```
 0013 OPARU

000000 i=0
```

```
03      000000 .dusr ?RUSP = i        ; ?USP word
04      000001 .dusr ?RUSL = i        ; User stack limit
05      000002 .dusr ?RUSB = i        ; User stack base
06      000003 .dusr ?RUST = i        ; User starting address
07      000004 .dusr ?RSII = i        ; Start of impure initialization area
08      000005 .dusr ?REII = i        ; End of impure initialization area
09      000006 .dusr ?RTMT = i        ; Highest available memory address
10      000007 .dusr ?RTSI = i        ; Start of user impure area
11      000010 .dusr ?RTEI = i        ; End of user impure area
12      000011 .dusr ?RTSP = i        ; Start of user pure area
13      000012 .dusr ?RTEP = i        ; End of user pure area
14      000013 .dusr ?RTLN = 1?n26
15
16
17
18

**00000 TOTAL ERRORS, 00000 PASS 1 ERRORS
  0014 OPARU



DUMO    177700      3/43#
FQ1     000000      2/03#
FQ2     000001      2/04#
FQ3     000002      2/05#
FQ4     000003      2/06#
L?N0    000004      2/07#   9/40    9/41    9/42    9/43
L?N1    000001      3/36#
L?N10   000015      7/37#
L?N11   000001      7/50#   8/03
L?N12   000016      8/04#
L?N13   000007      8/18#
L?N14   000034      9/43#   9/44    9/60
L?N15   000001      9/57#  10/01
L?N16   000042     10/05#
L?N17   000004     10/23#
L?N18   000006     10/39#
L?N19   000007     10/55#
L?N2    000001      3/45#
L?N20   000006     11/08#
L?N21   000006     11/23#
L?N22   000011     11/40#
L?N23   000005     11/54#
L?N24   000011     12/10#
```

```
L?N25     000005        12/24#
L?N26     000013        13/14#
L?N3      000001         4/02#
L?N4      000011         5/44#
L?N5      000011         6/04#   6/06
L?N6      000012         6/22#
L?N7      000004         6/37#



  0015 OPARU

L?N8      000001         7/01#   7/09
L?N9      000012         7/11#
RES1      001760         3/60#
RES2      077770         7/48#
?ATAT     000010         3/14#
?ATDC     000020         3/15#
?ATPM     000001         3/11#
?ATRD     000002         3/12#
?ATWR     000004         3/13#
?ATZR     000100         3/16#
?BLVL     000005         3/52#
?BPAR     000003         3/51#
?BRAT     000017         3/53#
?BSTP     000001         3/50#
?C0050    000000         4/26#
?C0075    000001         4/27#
?C0110    000002         4/28#
?C0150    000004         4/30#
?C0300    000005         4/31#
?C0600    000006         4/32#
?C1200    000007         4/33#
?C1345    000003         4/29#
?C15S     100000         4/05#
?C1800    000010         4/34#
?C192K    000017         4/41#
?C1S      040000         4/03#
?C2000    000011         4/35#
?C2400    000012         4/36#
?C2S      140000         4/04#
?C3600    000013         4/37#
?C4800    000014         4/38#
?C5BC     000000         4/11#
```

```
?C605     001000        3/29#
?C6BC     002000        4/12#
?C7200    000015        4/39#
?C7BC     004000        4/13#
?C8BC     006000        4/14#
?C8BT     000200        3/31#
?C9600    000016        4/40#
?CBIN     002000        3/28#
?CCK0     000011        4/19#
?CCK1     000017        4/20#
?CCK2     000000        4/21#
?CCK3     000001        4/22#
?CECH     010000        3/26#
?CEMM     000040        3/33#
?CESC     020000        3/25#
?CEVN     030000        4/09#
?CICC     000020        3/34#
?CLST     004000        3/27#
?CLVL     006000        3/59#
?CNAS     040000        3/24#
?CNED     000100        3/32#
?CNPR     000000        4/07#
?CODD     010000        4/08#
?CPAR     030000        3/58#
?CRAT     000017        4/01#
?CST      100000        3/23#
?CSTP     140000        3/57#

  0016 OPARU

?CUCI     000040        3/44#
?CUCO     000400        3/30#
?DAB      000003        7/05#
?DBBS     000015        3/04#   3/05
?DBPG     000004        2/55#   2/56
?DBRK     000011        2/60#   3/01
?DCHR     177773        2/38#
?DDIR     177775        2/40#
?DDVC     177774        2/39#
?DFB      000001        7/04#
?DIDF     000012        3/01#   3/02
?DIXF     000013        3/02#   3/03
?DLE1     040000        6/57#
?DLE2     020000        6/58#
?DLIB     000006        2/57#   2/58
```

```
?DLN     000012        7/11#
?DLNK    000014        3/03#    3/04
?DLOG    000100        3/06#
?DLPT    177772        2/37#
?DMBS    000016        3/05#
?DME1    010000        6/59#
?DME2    004000        6/60#
?DMSG    177776        2/41#
?DOBF    000000        2/51#    2/52
?DOLF    000003        2/54#    2/55
?DPRG    000002        2/53#    2/54
?DPSH    177777        2/42#
?DPST    000005        2/56#    2/57
?DRER    000010        7/09#
?DSMN    000000        2/44#    2/51
?DSMX    000100        2/45#    2/46     3/06
?DSP     000003        5/08#
?DSTF    000001        2/52#    2/53
?DSTW    000007        7/08#
?DTAL    000006        7/07#
?DTMX    000005        7/06#
?DTXT    000010        2/59#    2/60
?DUDF    000007        2/58#    2/59
?DUER    000011        7/10#
?DUMN    000101        2/46#
?DUMX    000200        2/47#
?DWRP    100000        6/56#
?EDMI    076000        2/17#    2/19
?EDMX    077777        2/18#
?EFLN    000001        7/50#
?ELN     000016        8/04#
?EPIP    000046        5/11#
?EVCH    076000        2/19#
?EVMAX   075777        2/16#
?EVMIN   000001        2/15#
?FATR    000002        6/17#
?FESZ    000003        6/18#
?FLEN    000010        6/21#
?FLN     000012        6/22#
?FSP     000002        5/07#
?FTLA    000004        6/19#
?FTLM    000006        6/20#
?FTYP    000001        6/16#
?GARG    000002        6/43#    6/44
```

```
   0017 OPARU

?GCMD     000000         6/41#   6/42
?GCNT     000001         6/42#   6/43
?GIDT     000001        12/21#
?GNUM     000001         6/34#
?GREQ     000000         6/33#
?GRES     000003         6/36#
?GSTP     000422         5/23#
?GSW      000002         6/35#
?GSWI     000005         6/46#
?GSWS     000004         6/45#   6/46
?GSYT     000003        12/22#
?GTLN     000004         6/37#
?GTSW     000003         6/44#   6/45
?HBUF     000002        11/19#
?HDLN     000011        11/40#
?HDTP     000417         5/20#
?HEAD     000005        11/22#
?HELN     000006        11/23#
?HETH     000001        11/32#
?HETL     000002        11/33#
?HIDH     000007        11/38#
?HIDL     000010        11/39#
?HLEN     000003        11/20#
?HSAD     000004        11/21#
?HSTH     000003        11/34#
?HSTL     000004        11/35#
?HSTP     000416         5/19#
?HTIK     000001        11/18#
?HUTH     000005        11/36#
?HUTL     000006        11/37#
?IDAT     000005        10/53#
?IHND     000001        10/49#
?IHPR     000006        10/54#
?IMAD     000002        10/21#
?IMLN     000003        10/22#
?IMSK     000002        10/50#
?INCH     000000         2/12#
?IOBC     000007        12/08#
?IOBT     000010        12/09#
?IOCH     000001        12/02#
?IOFP     000003        12/04#
?IOP      000421         5/22#
?IOSN     000002        12/03#
```

```
?IOSO    000006        12/07#
?IOSP    000005        12/06#
?IPCLN   000004        10/23#
?IRAD    000004        10/37#
?IRLN    000005        10/38#
?ISAD    000002        10/35#
?ISDL    000003        10/36#
?ISP     000004         5/09#
?ISRLN   000006        10/39#
?ISTK    000003        10/51#
?ISTL    000004        10/52#
?ITLM    000001        10/20#
?ITLN    000007        10/55#
?ITYP    000414         5/17#
?LDAT    000004        11/06#
?LDER    000004         4/51#

  0018 OPARU

?LEND    000002         4/50#
?LHND    000001        11/03#
?LHPR    000005        11/07#
?LSTK    000002        11/04#
?LSTL    000003        11/05#
?LSTR    000001         4/49#
?LTLN    000006        11/08#
?LTYP    000415         5/18#
?LXYZ    001777         4/52#
?MSNP    000004        11/53#
?MSP     000420         5/21#
?MSPP    000003        11/52#
?MSSN    000001        11/50#
?MSSP    000002        11/51#
?MXFL    000017         2/21#
?MXLD    000003         2/24#
?MXLL    000077         2/25#
?MXOV    000040         2/29#
?MXPL    000177         2/22#
?MXPR    000377         2/32#
?MXSL    000005         2/26#
?MXSP    000200         2/27#
?MXSW    000010         2/31#
?MXTK    000400         2/28#
?NMCH    000100         2/10#
?OACO    177774        12/43#
```

```
?OAC1    177775      12/44#
?OAC2    177776      12/45#
?OFP     177777      12/46#
?ORTN    000000      12/47#
?OUCH    000001       2/13#
?PASG    000013       8/01#
?PBIO    000005       7/57#
?PBLK    100000       7/47#
?PCHN    000011       7/59#
?PCIO    000007       7/58#
?PCPU    000003       7/56#
?PHMA    000007       5/42#
?PIMN    000003       5/38#
?PIMX    000004       5/39#
?PIP     000000       5/05#
?PLEV    000006       5/41#
?PLN     000011       5/44#
?POCH    000010       5/43#
?PPMN    000001       5/36#
?PPMX    000002       5/37#
?PPRI    000015       8/03#
?PREV    000005       5/40#
?PRLN    000015       7/37#
?PROT    000004       7/49#
?PRP     000045       5/10#
?PSTS    000014       8/02#
?PTIM    000001       7/55#
?PTSK    000012       7/60#
?RCH0    000003       7/27#
?RCH1    000004       7/28#
?RDIR    000006       7/30#
?REII    000005      13/08#
?RMAS    000013       7/35#

  0019 OPARU

?RMBP    000001       7/25#
?RMCH    000010       7/32#
?RMEM    000012       7/34#
?RMLN    000002       7/26#
?RMON    000014       7/36#
?RMTC    000011       7/33#
?RPRI    000007       7/31#
?RSII    000004      13/07#
?RSLI    000005       7/29#
```

?RTEI	000010	13/11#	
?RTEP	000012	13/13#	
?RTLN	000013	13/14#	
?RTMT	000006	13/09#	
?RTSI	000007	13/10#	
?RTSP	000011	13/12#	
?RUSB	000002	13/05#	
?RUSL	000001	13/04#	
?RUSP	000000	13/03#	
?RUST	000003	13/06#	
?SAC0	000001	9/29#	
?SAC1	000002	9/30#	
?SAC2	000003	9/31#	
?SAC3	000004	9/32#	
?SAF3	000030	9/42#	
?SCMN	000000	9/59#	
?SFA0	000014	9/39#	
?SFA1	000020	9/40#	
?SFA2	000024	9/41#	
?SFMP	000003	8/14#	
?SFPS	000012	9/38#	
?SGAS	100000	9/02#	
?SGB2	000002	8/34#	
?SGBK	000001	8/33#	
?SGBP	020000	8/30#	
?SGCB	000030	9/09#	
?SGCD	000003	8/35#	
?SGCE	000031	8/58#	9/10#
?SGCL	000006	8/43#	
?SGEX	000004	8/36#	
?SGFE	000021	8/57#	
?SGIO	000026	9/07#	
?SGIT	000027	9/08#	
?SGJO	000023	9/04#	
?SGKB	000036	9/15#	
?SGKL	000035	9/14#	
?SGOL	000005	8/37#	
?SGRB	000034	9/13#	
?SGRI	000032	9/11#	
?SGRT	000033	9/12#	
?SGSC	040000	8/41#	
?SGSO	000020	8/56#	
?SGUC	010000	8/54#	
?SGUS	000000	8/32#	
?SGVT	000024	9/05#	

```
?SGWP    000025      9/06#
?SIDS    000005      8/16#
?SIP     000047      5/12#
?SLN     000007      8/18#
?SMEM    000002      8/13#

  0020 OPARU

?SPCC    000005      9/33#
?SPNK    000036     10/02#
?SPRC    000004      8/15#
?SPRI    000035     10/01#
?SREV    000001      8/12#
?SRLN    000042     10/05#
?SRP     000412      5/15#
?SRTP    000410      5/13#
?SSBD    000006      8/17#
?SSFP    000007      9/35#
?SSSL    000011      9/37#
?SSSP    000006      9/34#
?STDR    040000      9/55#
?STKMIN  000062     12/37#
?STMO    000037     10/03#
?STOL    000041     10/04#
?STST    000034      9/60#
?STUS    020000      9/56#
?STWT    100000      9/54#
?SUSP    000010      9/36#
?SVNL    000017      2/23#
?SWLN    000034      9/44#
?SWTP    000411      5/14#
?S_RP    000413      5/16#
?TAC2    000006      6/01#
?TDP     000001      5/06#
?TKPP    000010      6/03#
?TLN     000011      6/06#
?TMP     000001     12/49#
?TPRI    000001      5/55#
?TSTA    000002      5/56#
?TSTB    000003      5/57#
?TSTE    000005      5/59#
?TSTL    000004      5/58#
?TUSP    000007      6/02#
?TYPE    000000      5/35#
?USP     000016     12/38#
```

```
0001 MERCO    MP/MASM Assembler    Rev 03.10  04/12/82 15:13:32
                 .title mercod

                ;========================
                ; ERCOD error codes
                ;========================





                ; start of error codes

        040001 .dusr ERNAR = 040001    ; *Argument does not exist
        040002 .dusr ERBTL = 040002    ; *Buffer too long
        040003 .dusr ERIRB = 040003    ; Buffer too short
        040004 .dusr ERPRM = 040004    ; Cannot delete permanent file
        040005 .dusr ERREN = 040005    ;*Renaming error (file is open, cross device)
        040006 .dusr ERDVC = 040006    ;*Invalid device code
        040007 .dusr ERDAI = 040007    ; Device is in use
        040010 .dusr ERDFT = 040010    ;*Fatal device error
        040011 .dusr ERDOL = 040011    ;*Device is off line
        040012 .dusr ERFIL = 040012    ; Device read error
        040013 .dusr ERPWL = 040013    ; Device write error
        040014 .dusr ERDID = 040014    ; Directory delete error
        040015 .dusr ERLAB = 040015    ;*Disk label does not match disk name
        040016 .dusr ERFIX = 040016    ; Disk requires fixup
        040017 .dusr EREOF = 040017    ; End of file
        040020 .dusr ERUIH = 040020    ;*Extant user interrupt handler
        040021 .dusr ERNAE = 040021    ; File already exists
        040022 .dusr ERFDE = 040022    ; File does not exist
        040023 .dusr EREOP = 040023    ;*File is in use
        040024 .dusr ERATP = 040024    ;*File is attribute protected
        040025 .dusr ERFTL = 040025    ; File name is too long
        040026 .dusr ERIFT = 040026    ; Illegal file type
        040027 .dusr ERIOO = 040027    ; Illegal option combination
        040030 .dusr ERSTS = 040030    ; Invalid stack definition (too small, system space)
        040031 .dusr ERSPC = 040031    ; Insufficient file space
        040032 .dusr ERMPR = 040032    ; Invalid address
        040033 .dusr ERMWT = 040033    ;*Multiple waiters for single NPSC
        040034 .dusr ERIAT = 040034    ;*Invalid attributes
```

```
040035 .dusr ERICN = 040035    ; Invalid channel number
040036 .dusr ERIFC = 040036    ; Invalid character in pathname
040037 .dusr ERICH = 040037    ;*Invalid characteristics
040040 .dusr EREVT = 040040    ;*Invalid event number ( > ?EVMAX or < ?EVMIN )
040041 .dusr ERMEM = 040041    ; Invalid memory request
040042 .dusr ERIOD = 040042    ;*Invalid operation for device
040043 .dusr ERPRP = 040043    ; Invalid priority
040044 .dusr ERADR = 040044    ; Invalid process address
040045 .dusr ERTID = 040045    ; Invalid task identifier
040046 .dusr ERLTL = 040046    ; Line is too long
040047 .dusr ERNDP = 040047    ;*No debugger present
040050 .dusr ERNMC = 040050    ; No free channels
040051 .dusr ERNOT = 040051    ; No free TCB available
040052 .dusr ERNUI = 040052    ;*No such user interrupt service routine exists
040053 .dusr ERNAD = 040053    ; Non-directory entry in pathname
040054 .dusr ERNSY = 040054    ;*Non-system name specified
040055 .dusr ERTMO = 040055    ;*Pend timeout
040056 .dusr ERANG = 040056    ;*Range error
040057 .dusr ERRAD = 040057    ; Read access denied
040060 .dusr ERSTL = 040060    ;*Searchlist overflow

0002 MERCO
                 ; or a value of 0 was on PROC packet
040061 .dusr ERNSS = 040061    ;*Switch not found
040062 .dusr ERTIP = 040062    ;*Task in progress
040063 .dusr ERWAD = 040063    ; Write access denied
040064 .dusr ERYSL = 040064    ;*Program internal error
040065 .dusr ERISC = 040065    ;*Illegal system call
040066 .dusr ERINT = 040066    ;*Internal error
040067 .dusr ERRNA = 040067    ;*No available resource
040070 .dusr ERCIN = 040070    ;*Console interrupt (^C^A)
040071 .dusr ERABT = 040071    ;*Son terminated via ^C^B
040072 .dusr ERIPT = 040072    ;*Illegal packet type
040073 .dusr ERPCA = 040073    ;*Call aborted due to program management call
040074 .dusr ERVNS = 040074    ; Program file format revision not supported
040075 .dusr ERDNM = 040075    ;*Device not mounted
040076 .dusr ERMLD = 040076    ;*Maximum link depth exceeded
040077 .dusr EROVN = 040077    ; Invalid overlay descriptor
040101 .dusr EREXS = 040101    ; Attempt to exceed maximum swap level
040102 .dusr ERNOV = 040102    ;*No overlays defined for this program
040103 .dusr EROVC = 040103    ;*Specified overlay is not currently in use
040104 .dusr ERATD = 040104    ;*All tasks have died
040105 .dusr ERUSD = 040105    ;*User and system debuggers can not coexist
040106 .dusr ERNEM = 040106    ; Not enough memory
040107 .dusr ERABK = 040107    ;*Son terminated via ^C^E
```

```
  040110 .dusr ERESZ = 040110    ;*Invalid element size
  040111 .dusr ERIFF = 040111    ;*Invalid file format (0bad SA file)
  040112 .dusr ERJMO = 040112    ;*User PC is equal to zero
  040113 .dusr ERSAD = 040113    ;*Scheduling already disabled (0from ?ERSCH,CK)
  040114 .dusr ERIWC = 040114    ;*Illegal word count (0 <2 )
  040115 .dusr ERBMT = 040115    ;*Bad or runaway tape, or format error
  040116 .dusr ERUDE = 040116    ;*Uncorrectable data error (0parity, etc)
  040117 .dusr ERFTH = 040117    ;*Fatal tape hardware error
  040120 .dusr EROCR = 040120    ;*Odd number of chars read from tape
  040121 .dusr ERTWL = 040121    ;*Tape write lock
  040122 .dusr ERIRN = 040122    ;*Illegal record number
  040123 .dusr ERIFN = 040123    ;*Illegal file number
  040124 .dusr ERLET = 040124    ; Logical EOT encountered
  040125 .dusr ERTNO = 040125    ;*Tape drive not open
  040126 .dusr ERPET = 040126    ;*Physical end of tape
  040127 .dusr ERBOT = 040127    ;*Unexpected beginning of tape
  040130 .dusr ERTMR = 040130    ;*Too many records in tape file (0 >65,563)
  040131 .dusr ERTFE = 040131    ; Tape format error
  040132 .dusr ERBRK = 040132    ; Comm. device break error
  040133 .dusr ERFRM = 040133    ; Comm. device framing error
  040134 .dusr ERPRT = 040134    ; Comm.. device parity error
  040135 .dusr ERORN = 040135    ; Comm. device receiver overrun
  040136 .dusr ERNSD = 040136 ; No such device

         ;
         ; Started adding MP/AOS specific error codes here.

         ;    Note:   The * is a MP/OS specific error code which has no
         ;                AOS counterpart.


         ;  MP/OS error classes returned on ?EXEC

  000000 .dusr ?ECCP = 0             ; code returned by called program
  000001 .dusr ?ECEX = 1             ; the error occurred while attempting the
                                       ; ?EXEC, the called program did not run
  000002 .dusr ?ECRT = 2             ; the error occurred on a ?RETURN which

 0003 MERCO
                                       ; did not complete, this error is seen
                                       ; by the grandparent of the program
                                       ; attempted the ?RETURN
  000003 .dusr ?ECBK = 3             ; the error occurred while trying to
                                       ; write a breakfile
  000004 .dusr ?ECAB = 4             ; the error returned indicates an
```

```
07                                          ; abnormal termination (0e.g. ^C^B) as
08                                          ; opposed to the usual ?RETURN
09
10

**00000 TOTAL ERRORS, 00000 PASS 1 ERRORS
  0004 MERCO

ERABK   040107     2/23#
ERABT   040071     2/10#
ERADR   040044     1/48#
ERANG   040056     1/58#
ERATD   040104     2/20#
ERATP   040024     1/32#
ERBMT   040115     2/29#
ERBOT   040127     2/39#
ERBRK   040132     2/42#
ERBTL   040002     1/14#
ERCIN   040070     2/09#
ERDAI   040007     1/19#
ERDFT   040010     1/20#
ERDID   040014     1/24#
ERDNM   040075     2/14#
ERDOL   040011     1/21#
ERDVC   040006     1/18#
EREOF   040017     1/27#
EREOP   040023     1/31#
ERESZ   040110     2/24#
EREVT   040040     1/44#
EREXS   040101     2/17#
ERFDE   040022     1/30#
ERFIL   040012     1/22#
ERFIX   040016     1/26#
ERFRM   040133     2/43#
ERFTH   040117     2/31#
ERFTL   040025     1/33#
ERIAT   040034     1/40#
ERICH   040037     1/43#
ERICN   040035     1/41#
ERIFC   040036     1/42#
ERIFF   040111     2/25#
ERIFN   040123     2/35#
ERIFT   040026     1/34#
ERINT   040066     2/07#
ERIOD   040042     1/46#
```

```
ERIOO    040027        1/35#
ERIPT    040072        2/11#
ERIRB    040003        1/15#
ERIRN    040122        2/34#
ERISC    040065        2/06#
ERIWC    040114        2/28#
ERJMO    040112        2/26#
ERLAB    040015        1/25#
ERLET    040124        2/36#
ERLTL    040046        1/50#
ERMEM    040041        1/45#
ERMLD    040076        2/15#
ERMPR    040032        1/38#
ERMWT    040033        1/39#
ERNAD    040053        1/55#
ERNAE    040021        1/29#
ERNAR    040001        1/13#
ERNDP    040047        1/51#
ERNEM    040106        2/22#
ERNMC    040050        1/52#
ERNOT    040051        1/53#
ERNOV    040102        2/18#

  0005 MERCO

ERNSD    040136        2/46#
ERNSS    040061        2/02#
ERNSY    040054        1/56#
ERNUI    040052        1/54#
EROCR    040120        2/32#
ERORN    040135        2/45#
EROVC    040103        2/19#
EROVN    040077        2/16#
ERPCA    040073        2/12#
ERPET    040126        2/38#
ERPRM    040004        1/16#
ERPRP    040043        1/47#
ERPRT    040134        2/44#
ERPWL    040013        1/23#
ERRAD    040057        1/59#
ERREN    040005        1/17#
ERRNA    040067        2/08#
ERSAD    040113        2/27#
ERSPC    040031        1/37#
ERSTL    040060        1/60#
```

ERSTS	040030	1/36#
ERTFE	040131	2/41#
ERTID	040045	1/49#
ERTIP	040062	2/03#
ERTMO	040055	1/57#
ERTMR	040130	2/40#
ERTNO	040125	2/37#
ERTWL	040121	2/33#
ERUDE	040116	2/30#
ERUIH	040020	1/28#
ERUSD	040105	2/21#
ERVNS	040074	2/13#
ERWAD	040063	2/04#
ERYSL	040064	2/05#
?ECAB	000004	3/06#
?ECBK	000003	3/04#
?ECCP	000000	2/57#
?ECEX	000001	2/58#
?ECRT	000002	2/60#

```
0001 OERCO    MP/MASM Assembler    Rev 03.10  04/12/82 15:13:54
                .title oercod
                ;


       053136 .dusr ERNPC = 053136    ;*no debugee process started
       053137 .dusr ERNSG = 053137    ;*no outstanding signal
       053140 .dusr ERNDB = 053140    ;*no process to debug
       053141 .dusr ERRLN = 053141    ;*invalid relative device number
       053142 .dusr ERMAP = 053142    ;*insufficient map slots
       053143 .dusr ersdb = 053143    ; fatal disk error in system data base
                                      ; (MDV or Label)
       053144 .dusr ERPID = 053144    ;invalid process identifier
       053145 .dusr ERKIL = 053145    ;on ?EXEC, process was killed by ?KILL
       053146 .dusr ERTSK = 053146    ;* on ?PROC invalid value for max # tasks
       053147 .dusr ERCHN = 053147    ;* on ?PROC, too many channels specified
       053150 .dusr ERISZ = 053150    ; Illegal segment size
       053151 .dusr ERNFS = 053151    ; No free segment
       053152 .dusr ERSNA = 053152    ; Segment is not attached
       053153 .dusr ERSAA = 053153    ; Segment is already attached
       053154 .dusr ERTMS = 053154    ; Too many segment attaches
       053155 .dusr ERSDE = 053155    ; Segment does not exist
       053156 .dusr ERIMA = 053156    ; Segment map area is not within 0-31
       053157 .dusr ERMLS = 053157    ; Request is longer than segment
       053160 .dusr ERRST = 053160   ; internal error indicating non-quiescent
                                     ;...path should be reset.
       053161 .dusr ERFRZ = 053161   ; internal errorindicating non-quiescent
                                     ;...path should be frozen.
       053162 .dusr ERNMF = 053162    ; No more fcbs are available
       053163 .dusr ersto = 053163    ; stack overflow
       053164 .dusr erfex = 053164    ; floating exception
       053165 .dusr ercme = 053165    ; commercial exception
       053166 .dusr ervtp = 053166    ; validity trap
       053167 .dusr erwpt = 053167    ; write protect trap
       053170 .dusr eriot = 053170    ; io protection trap
       053171 .dusr eritp = 053171    ; indirection protection trap
       053172 .dusr erari = 053172    ; alpha reserved instruction trap
       053173 .dusr ertad = 053173    ; invalid target address
       053174 .dusr errnf = 053174    ; resource not found (returned by
                                      ; q manipulation routines)
       053175 .dusr erdir = 053175    ; a value of 0 was specified on proc packet
       053176 .dusr erxqt = 053176    ; XQT of XQT
       053177 .dusr eridf = 053177    ; Out of idef DCTs
       053200 .dusr erhis = 053200    ; Already histogramming PID
       053201 .dusr erifp = 053201    ; FPU has previously been initialized
       053202 .dusr ernon = 053202    ; attempt to input more overlay nodes
                                      ; than user has allocated
```

```
47        053203 .dusr erslt = 053203    ; ?ALMP slot number error
48        053204 .dusr erdcm = 053204    ; ?ALMP request could not be filled
49        053205 .dusr erpor = 053205    ; invalid port
50        053206 .dusr ersmb = 053206    ; receive buffer too small
51        053207 .dusr ercbs = 053207    ; connection broken by server
52        053210 .dusr ercbc = 053210    ; connection broken by customer
53        053211 .dusr ersrv = 053211    ; invalid server
54        053212 .dusr ersve = 053212    ; server already exists
55        053213 .dusr ersvl = 053213    ; server limit exceeded
56        053214 .dusr ercxl = 053214    ; connection limit exceeded
57        053215 .dusr erisn = 053215    ; invalid server name format
58        053216 .dusr ersrn = 053216    ; server has been removed and has no connections
59        053217 .dusr ernbc = 053217    ; no broken connections
60        053220 .dusr erlgb = 053220    ; log buffer request error

 0002 OERCO
01        053221 .dusr ermtp = 053221    ; memory trap
02        053222 .dusr ersnu = 053222    ; slots not in use
03        053223 .dusr eridc = 053223    ; Invalid data channel option for uEclipse
04        053224 .dusr eriic = 053224    ; invalid system call at interrupt level
05
06               ; end of error codes
07
08

**00000 TOTAL ERRORS, 00000 PASS 1 ERRORS
  0003 OERCO

ERARI     053172        1/36#
ERCBC     053210        1/52#
ERCBS     053207        1/51#
ERCHN     053147        1/15#
ERCME     053165        1/31#
ERCXL     053214        1/56#
ERDCM     053204        1/48#
ERDIR     053175        1/40#
ERFEX     053164        1/30#
ERFRZ     053161        1/26#
ERHIS     053200        1/43#
ERIDC     053223        2/03#
ERIDF     053177        1/42#
ERIFP     053201        1/44#
ERIIC     053224        2/04#
ERIMA     053156        1/22#
ERIOT     053170        1/34#
```

ERISN	053215	1/57#
ERISZ	053150	1/16#
ERITP	053171	1/35#
ERKIL	053145	1/13#
ERLGB	053220	1/60#
ERMAP	053142	1/09#
ERMLS	053157	1/23#
ERMTP	053221	2/01#
ERNBC	053217	1/59#
ERNDB	053140	1/07#
ERNFS	053151	1/17#
ERNMF	053162	1/28#
ERNON	053202	1/45#
ERNPC	053136	1/05#
ERNSG	053137	1/06#
ERPID	053144	1/12#
ERPOR	053205	1/49#
ERRLN	053141	1/08#
ERRNF	053174	1/38#
ERRST	053160	1/24#
ERSAA	053153	1/19#
ERSDB	053143	1/10#
ERSDE	053155	1/21#
ERSLT	053203	1/47#
ERSMB	053206	1/50#
ERSNA	053152	1/18#
ERSNU	053222	2/02#
ERSRN	053216	1/58#
ERSRV	053211	1/53#
ERSTO	053163	1/29#
ERSVE	053212	1/54#
ERSVL	053213	1/55#
ERTAD	053173	1/37#
ERTMS	053154	1/20#
ERTSK	053146	1/14#
ERVTP	053166	1/32#
ERWPT	053167	1/33#
ERXQT	053176	1/41#

Using Overlays

F

Overlay Programming Considerations

Only pure code (specified by the assembler .NREL 1 directive) can be placed in an overlay. If any overlay object files contain impure code, the Binder places that code in the main program's impure area.

If a .ENTO directive is found in a routine that you do not bind into an overlay, then the Binder sets the overlay descriptor to −1. The ?OVLOD and ?OVREL system calls perform no action if they are called with a −1 descriptor. This means that you can wait until bind time to decide whether or not to put a routine in an overlay.

The system preloads each node with the code of its first overlay; therefore, these overlays will already be in main memory when your program begins to run.

When you use ?OVLOD to request an overlay that is already loaded, the system does *not* load a new copy.

An overlay routine can not call another overlay into the same node. Any attempt to do so will suspend task execution indefinitely.

To protect multitasked programs, the system maintains a *use count* for each overlay node. This count is incremented whenever a task executes a ?OVLOD for the node and decremented whenever a task executes a ?OVREL.

When a task requests an overlay, some other task may be using a different overlay in the same node. In this case, the requesting task is pended until the node's use count becomes zero. The system then loads the new overlay and unpends any tasks waiting for it. Multitasking is discussed in Chapter 8.

If your program is not multitasked, and you neglect to release an overlay, then the next ?OVLOD that requests a different overlay for the node will "hang" your program, that is, block it from execution indefinitely.

A program may contain up to 128 nodes; each node may have up to 256 overlays. Overlay mode space is allocated with a granularity of 256 words. The overlay area (all nodes) is allocated with a granularity of 1K word.

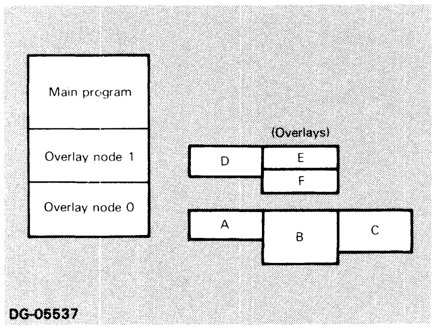


Figure F.1 Organization of sample program MPRG

To illustrate the use of overlays, we will discuss a typical program called MPRG. The programmer has decided that six subroutines are used infrequently and should therefore be placed in overlays. Some of these subroutines call each other, so two overlay nodes are needed. One node is to hold three subroutines, A, B and C, each in its own overlay. In the other node are two overlays: one contains subroutine D and another contains two subroutines, E and F. Figure F.1 illustrates the organization of the program MPRG.

Assembling Overlay Programs

To use overlays with a program, you must declare the names of the entry points of the overlay routines and the names of the overlays themselves. You use the assembler's .EXTN directive to declare all these names external symbols.

The overlays' names must be placed in your program's data area so that they can be referenced at run time. The Binder will replace the names with *overlay descriptors* used by the system calls.

The format of an overlay descriptor is shown in the following diagram:

*	NODE NUMBER	OVERLAY NUMBER
0	1 7	8 15

**Reserved for future use.*

For normal programming, there is no need for you to know this format; it is included here only for the sake of completeness.

The code for our sample program, MPRG, contains declarations in the following form:

```
      ;MAIN PROGRAM DECLARATIONS
     .EXTN OVL1,OVL2,OVL3,              ;Overlay descriptors
     .EXTN OVL4,OVL5                    ;Subroutine entries
     .EXTN A,B,C,D,E,F
 DESC1:   OVL1
 DESC2:   OVL2
 DESC3:   OVL3
 DESC4:   OVL4
 DESC5:   OVL5
.A:   A
.B:   B
.C:   C
.D:   D
.E:   E
.F:   F
```

Within the overlay source files, you must declare all the entry points with the .ENT directive and all the overlay names with the .ENTO directive.

Each of the six subroutines contains declarations in the following form:

```
 ;SUBROUTINE DECLARATIONS (subroutine A)
           .ENTO OVL1       ;Overlay descriptor
           .ENT A           ;Name of entry
           .
           .
           .
 A:                         ;(subroutine entry point)
           .
           .
           .
```

It is not necessary for you to explicitly allocate space for the overlay nodes. The Binder takes care of this.

Binding Overlay Programs

After assembling the main program and the overlays, you use the Binder to determine the actual distribution of nodes and overlays. The Binder then creates the program and overlay files.

Our sample program MPRG has seven object modules: MPRG.OB for the main program and A.OB, B.OB, etc. for the subroutines. The programmer binds the program using the command:

```
X BIND MPRG !* A ! B ! C *! !* D ! E F *!
```

This command contains special symbols defining the overlay structure to the Binder. The symbols **!*** and ***!** indicate the start and end of an overlay node, respectively. The symbol **!** defines separate overlays within a node. For instance, the string

```
!* D ! E F *!
```

identifies an overlay node with two overlays: one for D and another for E and F.

You must use delimiters (such as spaces) to separate the symbols **!**, **!***, and ***!** from each other and from the object program names.

The Binder analyzes the command and allocates space for the overlay nodes. Each node will be allocated enough memory to hold its largest overlay. The Binder then assigns values to the overlay descriptors and places these values in locations DESC1 through DESC5 of the main program. The Binder also resolves the references to the subroutine entries. The result of the binding process is a program file, MPRG.PR, and an overlay file, MPRG.OL.

For more information on binding overlays, refer to *MP/AOS Macroassembler and Binder Utilities* (DGC No. 069-400210).

Overlay System Calls

Two system calls support overlays: ?OVLOD and ?OVREL. You use the ?OVLOD call to load the overlay into its node. You then jump to the desired entry address. After exiting from the routine, you use the ?OVREL call to release the overlay.

The main program in our example must contain calls to manipulate the overlays, as the following example shows.

```
            ;MAIN PROGRAM OVERLAY CODE
MPRG:       ;Initialization.   (start of program)
LDA 0,OVL1  ;Get descriptor for routine.
?OVLOD      ;Load the overlay.
JMP ERROR   ;(Error return)
JSR @.A     ;Call the subroutine.
LDA 0,OVL1  ;Then set up for ?OVREL.
?OVREL      ;Release the overlay.
JMP ERROR   ;(Error return)
          . ;
          . ;
          . ;
```

MP/AOS Fatal and Booting Errors

G

Under some conditions the MP/AOS operating system detects an error from which it cannot recover. Such errors are called *fatal errors* and are extremely rare. Their most common cause is hardware malfunction. Non-fatal errors that the system detects can be reported using the ELOG (error logging) program discused in *System Generation and Related Utilities* (DGC No. 069-400206). See that document for a thorough discussion of the causes of both fatal and non-fatal errors.

Fatal Errors

Fatal errors can be either *kernel space* or *supervisor* errors. When the system detects a fatal error, it shuts down at once to prevent further data loss. At this time, depending on the type of fatal error, one of two messages is displayed on the console:

```
KERNEL PANIC
```

or

```
SUPERVISOR PANIC
```

Either of these messages is followed by a *fatal error code*, an octal number which identifies the cause of the error. A detailed list of fatal error codes and their meanings is included in your release package.

Following the fatal error code, the system displays additional information on the console, according to the error type encountered:

Kernel Errors

The fatal error code is followed by five octal numbers referring to the contents of the accumulators (AC0 through AC3) and the map status. Write down these numbers, as well as the error code, since they may assist you or Data General personnel in finding the cause of the error.

Supervisor Errors

The fatal error code is followed by an octal number representing the supervisor address which caused the error. As with kernel errors, write down the error code and the supervisor address to assist in identifying the cause of the error.

For a full analysis of the error, a dump of system memory, in combination with your symbol table files for the system, is necessary in addition to the information displayed by the system at the time the error occurred. Start the computer (via the Soft Console Panel, or the appropriate console switches, depending on your configuration) at location 14 octal. Instructions for taking the dump of system memory will then be displayed on your primary console.

Booting Errors

Booting errors may occur while booting a system from the console. When such errors are detected, the response to them occurs before the MP/AOS start-up message appears. The bootstrapping process halts and a message appears on the console:

```
Error =
```

followed by the *error code*, an octal number ranging from 0 through 4. This error code identifies the cause of the error; codes and their meanings are listed in Table G.1. When *Error = 0*, an additional number is printed after the zero; it indicates the status of the disk.

Code	Meaning
0	Disk error. Second number printed is the status of the disk.
1	Label block checksum error. You should boot another system disk and run FIXUP on the disk with the label block checksum error.
2	Checksum error while loading system (usually indicates a problem with memory).
3	No system installed on disk.
4	No FIXUP installed on disk.

Table G.1 MP/AOS booting errors

The Magnetic Tape Handler

H

MP/AOS supports magnetic tape devices as part of its library. To use the magnetic tape controllers, an MP/AOS program must first be bound with the tape routine library.

The library tape routines interface with the tape system by using the operating system's capability to support custom device handling routines. (See Chapter 10, User Device Support.) When you generate an MP/AOS system, respond to the SYSGEN query

```
Number of ?idef/?ldef device dcts?
```

by specifying one `?idef dct` (device control table) for every tape controller to be used. MP/AOS system generation is discussed in detail in *MP/AOS System Generation and Utilities* (DGC 069-400206).

The tape operations discussed in this appendix provide the MP/AOS system access to magnetic tape drives. (Figure H.1.) The DGC magnetic tape equipment handles the large reels of half-inch tape that are standard throughout the industry.

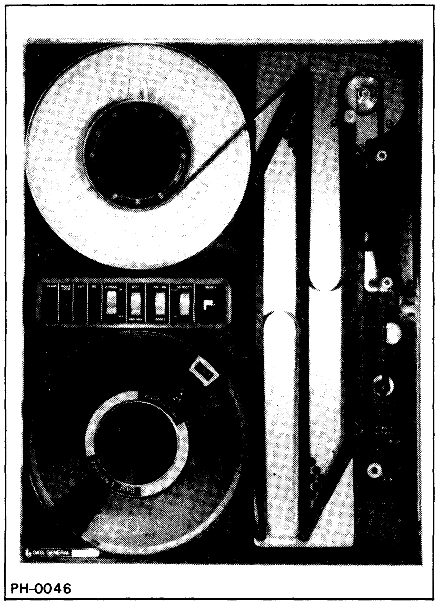

PH-0046

Figure H.1 DGC magnetic tape transport

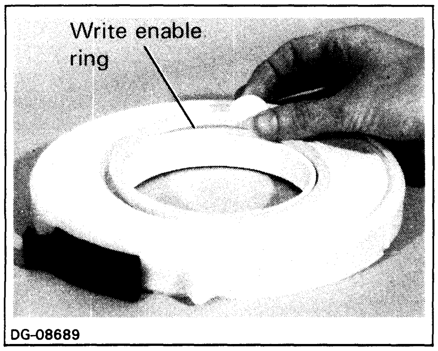
A tape system consists of a controller and up to eight tape drives. The 6021/6026 and the 6123/6125 controllers read and write tapes in two types of industry-compatible tape subsystems: i.e., NRZI or PE format.

Tape operations include reading from tape, writing to tape, moving tape to a new position and opening and closing the tape drive. Error recovery and return is also provided. Data transfers are of full two-byte words, grouped into records, which in turn may be grouped into files. Each byte transferred includes a parity bit which is used for error checks. Tape commands are used in the same way as system calls and library routines. They are presented in dictionary format at the end of this chapter.

Magnetic Tapes

The basic recording medium is a magnetic material coated on one side of a long half inch strip of tape usually made of mylar. The tape is held on large interchangeable reels which accommodate up to 2,400 feet per reel and are mounted on the supply hub of any conforming transport. When the transport is recording or reading information, the tape is moved from the supply reel past read/write heads to a take-up reel. As the tape moves, the heads define parallel data tracks along its surface. There are either seven or nine tracks on the tape; each track has both a read head and a write head.

Every tape has two physical markers indicating its extremities: the *loadpoint marker* and the *end-of-tape marker*. The markers are reflective strips sensed by photoelectric cells in the transport.



DG-08689

Figure H.2 Write enable ring

At least 10 feet in from the beginning of the reel is the loadpoint marker, which is the logical *beginning of the tape* (BOT). The transport automatically positions the tape at the BOT upon loading; reverse commands automatically stop at this marker. The BOT also provides an absolute reference point for all tape operations. A loadpoint gap of at least three inches precedes the first record on the tape.

The *end-of-tape marker* (EOT) is at least 14 feet from the physical end of the tape. A status bit indicates when the tape is beyond the EOT, but this condition stops the tape automatically only when it is moving forward.

An annular groove is molded into the back of every reel. The controller cannot write on the tape unless the supply reel has a plastic (write enable) ring in this groove. By removing the ring, the operator can protect the data on the tape from accidental destruction such as overwriting or erasure (Figure H.2).

Magnetic Tape Transports

DGC uses two industry-compatible systems for recording data on tape: Non-Return to Zero for Ones (NRZI) and Phase Encoded (PE). These systems format the tape and record data bits differently. NRZI tape drives operate on either a seven- or nine-track format, at 556 and 800 bits per inch (bpi) and in even or odd parity. PE tape drives operate on a nine-track format at 1600 bpi. Every transport accommodates two reels, one for supply and one for takeup.

In either type of subsystem, only one drive can be reading, writing or positioning the tape at any one time, but any number of drives can be rewinding simultaneously.

The Controllers

The 6021/6026 and the 6123/6125 controllers have device code 22, mnemonic MTA. The 6021/6026 control reads and writes tapes in both NRZI and PE formats, and it can move tape backward or forward to a new position. The 6123/6125 controller uses the PE recording method with a "streaming" mode tape drive for optimal performance.

Data Transfer

Data is stored as a "magnetic event" on the tape by the write head in the transport. As the tape moves past the write head, a sequence of data bits is written along the length of the tape. The number of data bits per inch (bpi) determines the data density for that transport.

Industry-compatible tape transports contain either seven or nine write heads, allowing simultaneous recording of a number of parallel tracks along the length of the tape. The data bits written simultaneously by a number of heads, one bit in each track, define a character on the tape. Each character, therefore, appears laterally, across the width of the tape.

A character is composed of a number of data bits and one parity bit used for error checking. Seven-track magnetic tape contains a six-bit byte of data and a parity bit in each character; a nine-track tape contains an eight-bit byte of data and a parity bit.

Transfers between memory and the controller are of full words of two bytes each, regardless of whether the tape contains six- or eight-bit bytes.

To write, the controller divides the words into data bytes and reassembles them when reading.

Depending on the particular transport, the data transfer rate ranges from 3,480 words per second to 36,000 words per second.

Records and Files

Data is grouped into *records,* composed of groups of words ranging in length from two to 4,096 words per record. The record is the smallest unit of information that can be addressed on tape.

To separate adjacent records, the controller automatically erases a segment of tape between them. The tape transport can only stop the tape in one of these inter-record gaps (IRG).

Records may be grouped together into *files.* Tape files, therefore, are groups of physically contiguous records; they should not be confused with disk files whose data are not necessarily contiguous.

The controller separates files from each other by an *end-of-file indicator* which is a three inch gap followed by an *end-of-file mark* — a special record containing a single, special data character and its longitudinal parity check character. The number of files which can be placed on a reel of tape depends on the length of the tape, the density of information, the number of words per record and the number of records per file.

Tape Operations

To run the tape, the program must select a transport and a command; all commands require the specification of parity. This information is passed by the program through AC2, as follows:

AC2 bits 13-15: drive number (0 - 7)
bit 9: parity (0 = odd, 1 = even)

Default parity is odd, and, for most operations, odd parity is desirable.

NOTE: *When writing in even parity, the program must take care not to supply a word containing a zero data byte in the recording format selected. This would result in a missing character (a blank line) which might be interpreted as an inter-record gap.*

Error Checking

The tape passes the read heads immediately after it passes the write heads. This allows a read-after-write system of error checking by means of a combination of lateral and longitudinal parity checks. The same combination of checks is also performed after a record is read.

Tape Commands

Preparation of a magnetic tape subsystem involves initializing the transport and positioning the tape. You can then issue the desired read or write commands. The remainder of this chapter discusses the routines for initializing, positioning, reading, writing and closing the drive.

The commands for these operations are used in the same manner as system calls and library routines. After assembly, the program must be bound with a tape routine library, MTA.LB for use under MP/AOS or MMTA.LB for use under AOS. For example:

```
BIND <your modules> MTA.LB
```

or

```
X MBIND/AOS <your modules> MMTA.LB
```

The following six routines are presented in dictionary format. For each entry in this chapter, we give the following information:

- the mnemonic that you place in your program code
- a description of the function performed
- tables summarizing inputs, outputs and errors. The contents of these tables are described below.

Inputs

This table lists information which your program must place in accumulators before executing the call.

Outputs

This table lists information which will be in the accumulators when control returns to your program.

Errors

This table lists the error codes likely to be returned if you use a call improperly. Error codes are returned in AC0.

For additional discussion of magnetic tapes, refer to *Programmer's Reference, Peripherals* (DGC No. 014-000632).

?TCLOS Close Tape Drive

Issues a rewind command to the specified drive and then removes it from the system. After the last open tape drive has been closed, a ?IRMV system call is performed, removing the tape controller from the system as well. The library routine invoked by this call also issues a ?DEMP system call to release the data channel map slots obtained by ?TOPEN.

Inputs

AC	Contents
AC2	Tape drive number (0-7), parity.

Outputs

None

Errors

Mnemonic	Meaning
ERTNO	Tape drive not open.
ERDOL	Device off line.

Get Current File and Record Number

?TGPOS

Gets the current file and record number of the specified tape drive.

Inputs

AC	Contents
AC2	Tape drive number (0-7), parity.

Outputs

AC	Contents
AC0	File number.
AC1	Record number.

Errors

Mnemonic	Meaning
ERTNO	Tape drive not open.

?TOPEN Open a Tape Drive

Introduces the specified drive to the system. If no previously opened drives are on the system, an ?IDEF system call is performed in order to set up the interrupt handler. The drive is then rewound and ready for use. The library routine invoked by this call also issues an ?ALMP system call for five data channel map slots on map A.

Inputs

AC	Contents
AC2	Tape drive number (0-7), parity.

Outputs

None

Errors

Mnemonic	Meaning
ERDOL	Device off line.
ERDAI	Device already in use.

Read a Record

?TREAD

Reads a record of *n* words,
2 <= *n* <= 4096
from the specified tape. If the currently positioned record is larger than *n* words, only *n* words are read, and the remainder is ignored. If the current record is less than *n* words, it is read in its entirety, but reading does not proceed beyond the record's end. The number of words actually read is always returned in AC1.

Inputs

AC	Contents
AC0	Address of the first word in memory to receive the data read.
AC1	Number of words to be read.
AC2	Tape drive number (0-7), parity.

Outputs

AC	Contents
AC0	Address of last word read, +1.
AC1	Number of words actually read.

Errors

Mnemonic	Meaning
ERTNO	Tape drive not open.
ERIWC	Illegal word count (<2).
EREOF	End of file encountered.
ERPET	Physical end of tape encountered.
EROCR	Odd number of characters read.
ERBMT	Bad or runaway tape, or format error.

?TSPOS Position Tape

Positions the tape on the specified drive immediately before the file or record to be accessed.

If the given file number is too large, the tape is positioned at the logical end of tape (EOT), and an error is returned.

If the given record number is too large, the tape is positioned at the end of the given file, and an error is returned.

When you specify
FILE -1/RECORD 0
as your file or record number, the tape is positioned at the logical EOT. This facilitates the process of appending to existing files or records. Use of any other negative numbers for either file or record will produce an error.

To rewind the tape, you specify
FILE 0/RECORD 0

Inputs

AC	Contents
AC0	File number.
AC1	Record number.
AC2	Tape drive number (0-7), parity.

Outputs

None

Errors

Mnemonic	Meaning
ERIFN	Illegal file number.
ERIRN	Illegal record number.
ERTNO	Tape drive not open.
ERPET	Physical end of tape encountered.
ERFTH	Fatal tape hardware error.
ERBOT	Unexpected beginning of tape.
ERTMR	Too many records in tape file (>65,536).

Write a Record to Tape

?TWRITE

Transfers n words from memory to the specified tape drive, when $n >= 2$. If $n = 0$, an EOF is written. If $n = -1$, a logical end-of-tape is written. If $n = 1$ or $n < -1$, no write operation is performed, and an error is returned.

While any record length between two and 4,096 words is legal, excessively short records will cause more tape to be used in the IRGs (inter-record gaps) than in the data stored. Very long records, on the other hand, make error recovery difficult. Hence, a record length of about 2 or 4K words is recommended for most applications.

If a record length of over 4,096 words is specified, only 4,096 words will actually be written, but the contents of AC1 will reflect this fact.

Inputs

AC	Contents
AC0	Address of first word in memory to be written.
AC1	Number of words to be written. If AC1 = 0, an EOF is written. If AC1 = −1, a logical end of tape (EOT) is written.
AC2	Tape drive number (0-7), parity.

Outputs

AC	Contents
AC1	Actual number of words written. Unchanged if original input to AC1 was 0 or −1.

Errors

Mnemonic	Meaning
ERTNO	Tape drive not open.
ERIWC	Illegal word count (<2).
ERTWL	Tape is write-locked.
ERPET	Physical end of tape encountered.
ERBMT	Bad or runaway tape, or format error.
ERFTH	Fatal tape hardware error.
ERUDE	Uncorrectable data error.

Running MP/AOS Programs Under AOS

I

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 to be used over the entire ECLIPSE computer line. This means, for example, that you might be able to write programs in SP/Pascal under AOS and run them on MP/AOS simply by rebinding.

NOTE: *For the mechanics of moving files to and from an MP/AOS disk, refer to the documentation of the following programs in MP/AOS File Utilities* (DGC No. 069-400204):

- AOSMIC, the AOS file transfer utility;
- the MOVE utility, if your system has magnetiic tape;
- the FOXFIRE file transfer utility, if you are transferring files over an asynchronous line.

The System Call Translator consists of three parts: the assembler's *permanent symbol table,* MASM.PS; a *translator object module,* MICREM.OB; and a *subroutine library file,* MMSL.LB.

MASM.PS, the assembler's permanent symbol table, has been prepared using the standard MP/AOS parameter files. You assemble your program with this file, instead of the usual MASM.PS. file. The translator object module and the subroutine library file contain preassembled code which translates your MP/AOS system calls into the AOS environment.

Assembling

To develop an MP/AOS program under AOS, you must assemble it using MMASM, the Macroassembler. You type the following CLI command:

X MMASM *pathname [...pathname]* <↵>

Each *pathname* represents the pathname of one or more files to be assembled.

NOTE: *Be sure that the MASM.PS that MMASM sees is the Translator's MASM.PS previously discussed.*

Binding

After assembly, you use the MP/AOS Binder to prepare an executable program file. Type the command:

X MBIND/AOS *pathname [...pathname]* <↵>

For this command to work properly, MICREM.OB, the translator object module, and MMSL.LB, the translator subroutine library file, must be on your searchlist, as well as the AOS user run-time library URT.LB. *Pathname* represents the pathname of one or more object files to be bound.

The result of this process is the program file, *progname*.PR, which can be run on an AOS system.

Translating a program prepared under AOS into an MP/AOS .PR file is a simple operation. Once the program has been assembled with the Translator's MASM.PS, a rebind is all that is required. The bind command is as follows:

X MBIND/MPAOS *pathname [...pathname]* <↵>

As before, *pathname* represents the pathname of one or more object files to be bound.

For more information refer to the Binder section in *MP/AOS Macroassembler, Binder, and Library Utilities* (DGC No. 069-400210).

Compatibility of System Calls

Since the AOS environment differs from that of MP/AOS, there are some differences in the actions of some system calls. These are detailed in the following paragraphs.

The translator converts AOS error codes into their MP/AOS counterparts. If the error code has no MP/AOS counterpart, your program receives the AOS code. You should be aware that a different error may be returned by the Call Translator than by MP/AOS.

Program Management

The ?RETURN call with the BK option uses the AOS convention for the break file name:
?*pid.time*.BRK
where *pid* is your process I.D. and *time* is the current time of day. Also, a program terminated with a ?RETURN BK may not pass a message to the parent program.

The ?EXEC call, where the complete pathname given to ?EXEC is CLI.PR, invokes AOS CLI with the appropriate message format. The user's message must be in the MP/AOS CLI format with the CLI as the zero argument.

NOTE: *Attempting to ?EXEC any program named CLI.PR other than AOS CLI.PR will cause the program to fail on start-up.*

Due to AOS restrictions, some or all the messages passed by ?EXEC will be capitalized.

An ?EXECuted program will fail if it attempts to use an exclusively opened channel passed from the parent.

The ?BOOT call does not perform a bootstrap. It attempts to return to the user's CLI no matter how many levels down that CLI is. The message gives the reason for returning and the name of the specified bootstrap device or file in one of two forms:

MP Emulator shutdown

MP Emulator booting: <file name>

Under the Call Translator, ?ERMSG reads from MERMES, which must, therefore, be locatable through the searchlist.

Unlike MP/AOS the Call Translator's handling of overlays does not use a channel internally. You should also be aware that overlay node sizing is larger under the Call Translator. Hence, programs with many nodes which fit under MP/AOS may not fit under the Call Translator. Further, under the Call Translator the first overlay in any node is *not* preloaded by the system; you must therefore issue an ?OVLOD to load it.

Multitasking

The Call Translator limits a program to 30 tasks.

The allocation scheme for task identifiers used under the Call Translator is unrelated to that used under MP/AOS.

Avoid running tasks at priority zero (0), since such tasks will compete with Call Translator tasks running at priority zero and may cause them to function incorrectly.

Task scheduling under the Call Translator is somewhat different than under MP/AOS. Hence, care should be taken to force scheduling of tasks through ?PEND and ?DRSCH calls and through setting different priorities.

The ?PEND call for a CTRL-C CTRL-A works only for @TTIO.

File Management

AOS supports file type numbers which are somewhat different from MP/AOS file types. The System Call Translator converts MP/AOS file types to their AOS counterparts when you create files. It also converts AOS file types to their MP/AOS counterparts when you open files created by AOS programs. The correspondences between file types are summarized in Tables I.1 and I.2.

MP/AOS	AOS	Meaning
?DDIR	?FDIR	Directory
?DIDF	?DMAX-1	MP/ISAM data file
?DIXF	?DMAX	MP/ISAM index file
?DLNK	?FLNK	Link
?DMBS	?DMAX-2	MP/BASIC save file
?DPRG	?FPRG	Program file
?DTXT	?FTXT	Text file
?DUDF	?FUDF	User data file

Table I.1 Conversions of MP/AOS file types when creating files under AOS

NOTE: *All file types not mentioned in the table above are converted to ?FUDF.*

AOS	MP/AOS	Meaning
?DMAX	?DIXF	MP/ISAM index file
?DMAX-1	?DIDF	MP/ISAM data file
?DMAX-2	?DMBS	MP/BASIC save file
?FCPD	?DDIR	AOS control point directory
?FDIR	?DDIR	Directory
?FDKU	?DDVC	Disk unit
?FGFN	?DCHR	AOS generic file
?FLNK	?DLNK	Link
?FLPU	?DLPT	Line printer
?FPRG	?DPRG	Program file
?FTXT	?DTXT	Text file
?FUDF	?DUDF	User data file

Table I.2 Conversions of AOS file types when opening files with MP/AOS programs

NOTE: *All file types not mentioned in the table above are converted to ?DUDF.*

Under AOS the system call ?RENAME is not supported across directories.

When you use the ?OPEN call with a CR (create) option, the System Call Translator does not use the element size supplied with the call. Instead, the default element size one is used.

No more than three non-pended calls may run concurrently. The user must specify additional TCB's (task control blocks) for calls non-pended.

Under the Call Translator, the searchlist has a maximum length of 511 characters and no error is produced if it contains more than five pathnames.

Due to AOS restrictions, the Call Translator allows no more than eight directory tree levels.

The call ?FSTAT CH on a disk unit where the channel is exclusively open may return an incorrect file length.

File attributes are also handled differently on the two systems. The MP/AOS System Call Translator intercepts the references in your program to all file attributes except permanence and translates them into elements on the *access control list* (ACL) of the file. The ACL is a file protection feature provided by AOS, which is described fully in the *Advanced Operating System (AOS) Programmer's Manual* (DGC No. 093-000120-03).

The correspondences between attributes and access types are summarized in Table I.3.

NOTE: *There is a reversal in polarity between the two systems: setting the MP/AOS read protect attribute for a file means that it may* not *be read; i.e, setting this attribute is tantamount to removing the AOS read access privilege (R). Conversely, setting the AOS R (read access privilege) for a file means that it* may *be read. (This conversion is handled by the Translator.)*

MP/AOS Attributes		**AOS Access Privileges**
Read protection:	may *not* be read	R : read access
Write protection:	may *not* be written	W : write access
Attribute protection:	may *not* change attributes	O : owner access

Table I.3 Reversal in polarity between MP/AOS attributes and AOS access privileges

The permanence attribute is handled identically under the AOS and MP/AOS systems.

I/O Device Management

The AOS and MP/AOS systems have different formats for device characteristics. The ?GCHAR and ?SCHAR calls perform the conversion between characteristics, so that the difference is transparent to your program. Note, however, that if you use the HC option with ?GCHAR or ?SCHAR, the following occurs: the ?GCHAR call returns a zero; the ?SCHAR call executes successfully, but ignores the HC option.

Special caution is also in order when you use the ?GCHAR and ?SCHAR calls with @TTI and @TTO. Refer to discussion following Table I.5. ?SCHAR with the LL option and a line length set to -1 allows only 256 characters per line, the maximum line length under AOS.

Table I.4 summarizes the correspondences between device characteristics for AOS and MP/AOS.

MP/AOS Name	AOS Name
?CBIN	Supported for @TTI, @TTO, @TTI1, @TTO1, @LPT
?CECH	?CEOC
?CEMM	?CEOS
?CESC	?CESC
?CICC	Not supported
?CLST	Not supported
?CNAS	?CNAS
?CNED	Supported for @TTI, @TTO, @TTI1, @TTO1, @LPT
?CST	?CST
?CUCO	?CUCO
?C605	?C605
?C8BT	Supported for @TTI, @TTO, @TTI1, @TTO1, @LPT

Table I.4 Correspondences between device characteristics

?DSTAT does not store information in the packet. It simply validates its input parameters and then exits.

Tables I.5 and I.6 list the calls and library routines supported by the Call Translator. The MP/AOS calls listed in Table I.7 are not supported by the Call Translator and produce an error return with code ERISC (*illegal system call*) when attempted.

?ALIST
?AWAIT
?BOOT
?CLOSE
?CREATE
?CTASK
?DELETE
?DIR
?DRSCH
?DSTAT
?ERSCH
?EXEC
?FSTAT
?GCHAR
?GLIST
?GNAME
?GPOS
?GTATR
?GTIME
?GTMSG
?INFO
?KTASK
?MEMI
?MYID
?OPEN
?PEND
?PRI
?READ
?RENAME
?RESET
?RETURN
?SCHAR
?SCHS
?SPOS
?STATR
?STIME
?UNPEND
?WRITE

Table I.5 MP/AOS system calls supported under MP Emulator

?CDAY

?CTOD

?DELAY

?ERMSG

?FDAY

?FTOD

?GDAY

?GNFN

?GTOD

?MSEC

?OVLOD

?OVREL

?PCLOS

?POPEN

?PWRIT

?SLIST

?SDAY

?STOD

?TCLOS

?TGPOS

?TMSG

?TOPEN

?TREAD

?TSPOS

?TWRITE

Table I.6 MP/AOS library routines supported under the MP Emulator

?ALMP

?ASEG

?BLOCK

?CLEAR

?CSEG

?DCLR

?DEMP

?DHIS

?DISMOUNT

?DSBL

?DSEG

?EHIS

?EINFO

?ENBL

?EQT

?GIDS

?GMRP

?GTPID

?IDEF

?IFPU

?IPEND

?IRMV

?IUNPEND

?IXIT

?KILL

?LDEF

?LKUP

?LRMV

?LXIT

?MOUNT

?MSEG

?OBITS

?PROC

?PURGE

?RCV

?RCVA

?RDMEM

?RDST

?RMVE

?SD.R

?SEND

?SINFO

?STMP

?TPORT

?UNBLOCK

?WRMEM

?WRST

?WSIG

Table I.7 MP/AOS system calls not supported under MP Emulator

There is mapping between the MP/AOS system and AOS for the various devices, shown in Table I.8. The System Call Translator recognizes the MP/AOS device name and converts it to its AOS counterpart.

MP/AOS Device Name	AOS Device Name
@TTI	@Input (or @null if the program is batched)
@TTO	@Output
@TTI1	@Data
@TTO1	@List

Table I.8 Device name mapping

Because the characteristics for @TTI and @TTO both map into the generic AOS device @CONSOLE, you should exercise caution when using the calls ?GCHAR and ?SCHAR.

Setting any of the characteristics usable by both input and output on @TTI will also affect @TTO, and vice versa. In particular, the following sequence will cause problems.

```
?GCHAR @TTO
?SCHAR @TTO ; new characteristics
?GCHAR @TTI
?SCHAR @TTI ; new characteristics
.
.
.
?SCHAR @TTO ; restore characteristics
?SCHAR @TTI ; restore characteristics
```

This sequence will not restore characteristics properly, since the ?SCHAR @TTO call changes some of the characteristics of @TTI before the ?GCHAR @TTI saves them.

Instead, use this sequence for both @TTI and @TTO:

```
?GCHAR @TTO
?GCHAR @TTI
?SCHAR @TTO
?SCHAR @TTI
.
.
.
?SCHAR @TTO ; restore characteristics
?SCHAR @TTI ; restore characteristics
```

MP/AOS Library Routines

J

MP/AOS offers three libraries of subroutines. These are OSL.LB, a library of miscellaneous routines, MTA.LB, the magnetic tape library, and DCLP.LB, a library of routines for accessing the data channel line printer.

The OSL.LB library is automatically included in your BIND line, but you must specifically list either the magnetic tape or the data channel line printer libraries MTA.LB and DCLP.LB if you wish them bound with your program.

Table J.1 lists the routines in library OSL.LB. Tables J.2 and J.3 list the routines in libraries MTA.LB and DCLP.LB, respectively.

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 a date to internal format
?FTOD	Convert a time to internal format
?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 J.1 List of routines in library OSL.LB

Routine	Function
?TCLOS	Close tape drive
?TGPOS	Get current file and record number
?TOPEN	Open tape drive
?TREAD	Read a record
?TSPOS	Position tape
?TWRITE	Write a record to tape

Table J.2 List of routines in library MTA.LB

Routine	Function
?PCLOS	Close line printer
?POPEN	Open line printer
?PWRIT	Write to line printer

Table J.3 List of routines in library DCLP.LB

Sample IPC Programs

K

This appendix contains listings of two sample interprocessor communications (IPC) programs, SERV.LS and CUST.LS.

```
0001 SERV  MP/MASM ASSEMBLER REV 03.00  11/12/81 13:20:42
                     .title serv
                     .nrel
             ; Sample server program.  This program provides a server, 'noblanks',
             ; which accepts character strings, removes the blanks from them, and
             ; sends them back to the customer.

00000'020471 serv:   lda     0,.sv           ; byte pointer to server name
                     ?dclr                   ; declare self as server
00003'000461         jmp     err
00004'044463         sta     1,snum          ; save server number
00005'121000         mov     1,0             ; server number to ac0 for ?rcva
00006'070471         lef     2,rpack         ; wait for service request
                     ?rcva
00011'000453         jmp     err
00012'040456         sta     0,port          ; port for this customer
00013'044473         sta     1,len           ; length of his message
00014'044473         sta     1,count         ; length of his message
```

```
00015'060001         lef     0,?ouch         ; display received string in output
00016'024534         lda     1,.mess         ; file
00017'070011         lef     2,11
                     ?write
00022'000442         jmp     err
00023'024526         lda     1,.recbf
00024'030463         lda     2,count
                     ?write
00027'000435         jmp     err

              ; Remove blanks from input string and copy it to message buffer for ?send.
00030'030521         lda     2,.recbf        ; byte pointer to original message
00031'034517         lda     3,.senbf        ; byte pointer to send buffer
00032'024526         lda     1,blank         ; ascii blank
00033'142710 loop:   ldb     2,0             ; get byte from original string
00034'106414         sub#    0,1,szr         ; is it a blank?
00035'000404         jmp     notbl           ; no
00036'014450         dsz     len             ; yes...one less char to return
00037'000401         jmp     .+1
00040'000403         jmp     next            ; process next character

00041'163010 notbl:  stb     3,0             ; not a blank...put in send buffer
00042'175400         inc     3,3             ; point to next byte in send buffer
00043'151400 next:   inc     2,2             ; point to next byte of message
00044'014443         dsz     count           ; one less byte to scan
00045'000766         jmp     loop            ; process next character

00046'070435         lef     2,spack         ; done removing blanks
00047'020421         lda     0,port          ; send processed string to customer
                     ?send
00052'000412         jmp     err

              ; This server quits after processing one message.
00053'020414         lda     0,snum          ; server number
                     ?rmve                   ; remove server name
00056'000406         jmp     err
00057'020411         lda     0,port          ; clear connection with customer
                     ?clear
00062'000402         jmp     err
00063'102400         sub     0,0             ; and return
00064'152400 err:    sub     2,2
                     ?return


 0002 SERV
```

```

00067'000000 snum:   0                        ; server number
00070'000000 port:   0                        ; port number for current customer
00071'000164".sv:    sv*2                     ; byte pointer to server name
00072'067157 sv:     .txt 'noblanks<0>'       ; server name
      061154
      060556
      065563
      000000

00077'000004 rpack:  .blk    ?ipcln           ; ?rcva packet
      000103'        .push
      000077'        .loc    rpack+?type
00077'000412         ?srp                     ; packet type
      000100'        .loc    rpack+?itlm
00100'177777         -1                       ; no time limit
      000101'        .loc    rpack+?imad
00101'000260"        recbf*2                  ; byte pointer to receive buffer
      000102'        .loc    rpack+?imln
00102'000040         40                       ; byte length of buffer
      000103'        .loc    .pop

00103'000004 spack:  .blk    ?ipcln           ; ?send packet
      000107'        .push
      000103'        .loc    spack+?type
00103'000412         ?srp                     ; packet type
      000104'        .loc    spack+?itlm
00104'000002         2                        ; time limit is 2 seconds
      000105'        .loc    spack+?imad
00105'000220"        senbf*2                  ; byte pointer to message
      000106'        .loc    spack+?imln
00106'000000 len:    0                        ; number of bytes to send to customer
      000107'        .loc    .pop

00107'000000 count:  0                        ; number of bytes left to process
00110'000020 senbf:  .blk    20               ; message buffer for ?send
00130'000020 recbf:  .blk    20               ; buffer for ?rcva
00150'000220".senbf: senbf*2                  ; byte pointer to buffer for ?send
00151'000260".recbf: recbf*2                  ; byte pointer to buffer for ?rcva
00152'000326".mess:  mess*2                   ; byte pointer to header string
00153'046545 mess:   .txt    "Message:<12>"   ; header string
      071563
      060547
      062472
      005000
```

```
46 00160'000040 blank:  .txt    "<0> "          ; ascii blank
47       000000
48
49 00162'000200 stk:    200                     ; stack definition
50       000040         .loc    40
51 00040 000162'        stk
52       000042         .loc    42
53 00042 000342'        stk+160
54                      .end    serv
55

**00000 TOTAL ERRORS, 00000 PASS 1 ERRORS
  0003 SERV

BLANK 000160'       1/31    2/46#
COUNT 000107'       1/17    1/24    1/42    2/35#
ERR   000064'       1/09    1/14    1/22    1/26    1/48    1/53    1/56
                    1/58#
LEN   000106'       1/16    1/35    2/32#
LOOP  000033'       1/32#   1/43
MESS  000153'       2/40    2/41#
NEXT  000043'       1/37    1/41#
NOTBL 000041'       1/34    1/39#
PORT  000070'       1/15    1/46    1/54    2/03#
RECBF 000130'       2/18    2/37#   2/39
RPACK 000077'       1/12    2/11#   2/13    2/15    2/17    2/19
SENBF 000110'       2/30    2/36#   2/38
SERV  000000'       1/07#   2/54
SNUM  000067'       1/10    1/51    2/02#
SPACK 000103'       1/45    2/23#   2/25    2/27    2/29    2/31
STK   000162'       2/49#   2/51    2/53
SV    000072'       2/04    2/05#
.MESS 000152'       1/19    2/40#
.RECB 000151'       1/23    1/29    2/39#
.SENB 000150'       1/30    2/38#
.SV   000071'       1/07    2/04#
?CLEA 004577! MC    1/55
?DCLR 004636! MC    1/08
?I    000013        1/09#   1/14#   1/22#   1/26#   1/48#   1/53#   1/56#
                    1/60#
?J    000000        1/09#   1/14#   1/22#   1/26#   1/48#   1/53#   1/56#
                    1/60#
?RCVA 004675! MC    1/13
?RETU 000307! MC    1/59
?RMVE 005032! MC    1/52
```

```
?SCAL 000001$ XN    1/09    1/14    1/22    1/26    1/48    1/53    1/56
                    1/60
?SEND 004442! MC    1/47
?WRIT 002112! MC    1/21    1/25


0001 CUST  MP/MASM ASSEMBLER REV 03.00  11/12/81 13:20:16
                        .title cust
02                      .nrel
03              ; Sample customer program.  This program makes use of a server, 'noblanks',
04              ; which takes a character string and returns the same string with all
05              ; blanks removed.
06
07 00000'020430 cust:   lda     0,.sv           ; byte pointer to server name
08                      ?lkup                   ; get port to server
09 00003'000421         jmp     err
10 00004'040423         sta     0,port          ; remember port number
11
12 00005'070431         lef     2,pack          ; send string to server and
13                      ?sd.r                   ; wait for reply
14 00010'000414         jmp     err
15
16 00011'131000         mov     1,2             ; returned message length to ac2
17 00012'060001         lef     0,?ouch         ; write blankless string to output
18 00013'024442         lda     1,.recbf        ; file
19                      ?write
20 00016'000406         jmp     err
21 00017'020410         lda     0,port          ; all done with server, so break
22                      ?clear                  ; connection
23 00022'000402         jmp     err
24 00023'102400         sub     0,0             ; and return
25 00024'152400 err:    sub     2,2
26                      ?return
27
28
29
30
31 00027'000000 port:   0                       ; ipc port
32 00030'000062".sv:    sv*2                    ; byte pointer to server name
33 00031'067157 sv:     .txt 'noblanks<0>'      ; server name
```

```
34        061154
35        060556
36        065563
37        000000
38 00036'000006 pack:  .blk    ?isrln          ; packet for ?sd.r
39        000044'      .push
40        000036'      .loc    pack+?type
41 00036'000413        ?s_rp                   ; packet type
42        000037'      .loc    pack+?itlm
43 00037'000005        5                       ; time limit
44        000040'      .loc    pack+?isad
45 00040'000110"       senbf*2                 ; byte pointer to message
46        000041'      .loc    pack+?isln
47 00041'000020        20                      ; message byte length
48        000042'      .loc    pack+?irad
49 00042'000134"       recbf*2                 ; byte pointer to buffer for returned message
50        000043'      .loc    pack+?irln
51 00043'000030        30                      ; byte length of buffer
52        000044'      .loc    .pop
53 00044'044440 senbf:  .txt   "I am a message.<12>"   ; message
54        060555
55        020141
56        020155
57        062563
58        071541
59        063545
60        027012

 0002 CUST
01        000000
02 00055'000134".recbf: recbf*2                ; receive buffer byte pointer
03 00056'000014 recbf:  .blk    14             ; receive buffer
04
05
06
07 00072'000200 stk:   .blk 200                ; stack definition
08        000040       .loc    40
09 00040 000072'       stk
10        000042       .loc    42
11 00042 000252'       stk+160
12                     .end    cust
13
 ?I    000013        1/09#   1/14#   1/20#   1/23#   1/27#
?J    000000        1/09#   1/14#   1/20#   1/23#   1/27#
?LKUP 004403! MC    1/08
?RETU 000307! MC    1/26
?SCAL 000001$ XN    1/09    1/14    1/20    1/23    1/27
?SD.R 004540! MC    1/13
?WRIT 002112! MC    1/19
```

```
**00000 TOTAL ERRORS, 00000 PASS 1 ERRORS
  0003 CUST

CUST  000000'       1/07#   2/12
ERR   000024'       1/09    1/14    1/20    1/23    1/25#
PACK  000036'       1/12    1/38#   1/40    1/42    1/44    1/46    1/48
                    1/50
PORT  000027'       1/10    1/21    1/31#
RECBF 000056'       1/49    2/02    2/03#
SENBF 000044'       1/45    1/53#
STK   000072'       2/07#   2/09    2/11
SV    000031'       1/32    1/33#
.RECB 000055'       1/18    2/02#
.SV   000030'       1/07    1/32#
?CLEA 004577! MC    1/22
```

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

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

Data General

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

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. 093-400051-01

Yes	No			
□	□	Is this manual easy to read?	○ You (can, cannot) find things easily. ○ Language (is, is not) appropriate. ○ Technical terms (are, are not) defined as needed.	○ Other:
		In what ways do you find this manual useful?	○ Learning to use the equipment ○ As a reference ○ As an introduction to the product	○ To instruct a class. ○ Other:
□	□	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: __________

CUT ALONG DOTTED LINE

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 (F-131)
4400 Computer Drive
Westboro, MA 01581

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

CUT ALONG DOTTED LINE

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:

DataGeneral

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