M/2

System Description

Preliminary

Dansk Data Elektronik ApS

2 April 1982

Author: Claus Tøndering

M/2 System Description

Index

1. Introduction.

This manual contains a preliminary description of the M/2 operating system running on the SPC/2 computer. It describes the various concepts and system directives available, but does not include the utility program which will be available.

Numbers starting with a ´$´ character are hexadecimal, other numbers are decimal.

2. System Overview.

The SPC/2 computer consists of:

- One or more MC68000 micro processors with associated memory. These are termed the ´master CPUs´. These processors perform the actual execution of the user programs.

- A number of 8085 micro processors with associated memory. These are termed the I/O controllers. These processors are dedicated to the handling of one or more I/O devices.

- A block of memory common to all processors in the computer.

2.1. The Master CPUs.

Each master CPU is independent of the other master CPUs. Programs executing on one master CPU does so independently of programs executing on another master CPU, except for the synchronization of access to I/O controllers. There is no hierachy between the master CPUs, they are identical in their functioning.

The master CPUs are numbered 1, 2, etc.

Each master CPU consists of an MC68000 micro processor plus up to 16 megabyte of RAM memory.

2.2. The I/O Controllers.

Each I/O controller is dedicated to the handling of one or more I/O devices. The master CPUs perform no input/output themselves, instead they communicate with the I/O controllers and request these to perform the I/O desired. One I/O controller may serve several master CPUs simultaneously.

Each I/O controller consists of an 8085 micro processor plus up to 64 kilobyte of RAM memory.

The user cannot use the I/O controllers for executing programs.

2.3. The Common Memory.

A block of memory is common to all CPUs in the computer. This memory is used for synchronization and communication between the CPUs and cannot be accessed by the user.

2.4. M/2.

M/2 is the operating system servicing the SPC/2 computer. M/2 consists of software to handle the different CPUs in the computer.

Each master CPU is equipped with an identical copy of the M/2 software package designed to service master CPUs.

Each I/O controller is equipped with a copy of the M/2 software package designed to service that particular type of I/O controller.

When not explicitly stated otherwise, the term M/2 will be used to designate the operating system in the master CPUs.

M/2 is a multi-user multi-tasking operating system. Several users operating on the same SPC/2 may be executing their programs on the same master CPU or on diffent master CPUs without being aware of the difference, except, perhaps, for the execution speed. M/2 also supports inter-computer communication via the so-called MikNet, so a user may, for example, access a file in another computer in the same manner as a file in the computer to which the user is connected.

3. Bootstrapping the Computer.

One disk in the computer is termed the ´boot disk´. This is the disk, from which the operating system will be loaded when power is applied to the computer.

The boot disk must have a ´MIKFILE´ file system structure.

On the boot disk a number of files will be present. Some of these files contain self-test programs for the various CPUs in the computer, other files contain the actual operating system.

>> The names and detailed use of these files have not yet been determined <<

3.1. The Configuration File.

On the boot disk a ´configuration file´ must be present. This file has the name ´CONFIGUR.K´. This file contains the following information:

- The numbering and the types of the disks connected to the computer.
- The numbering and the types of the terminals connected to the computer.
- The numbering and the types of the printers connected to the computer.
- Various operating system parameters.

>> The format of this file has not yet been determined <<

It is possible to change the contents of the configuration file and thus alter the computer configuration.

4. Gaining Access to the System.

When the computer has been bootstrapped, all terminals, except number 1, will display the following:

```
M/2 version dd.mm.yyyy.

M/2 is ready for log on. Please press the escape key:
```

This is an invitation to the user to log on to the computer, that is, to acquire access to a master CPU. But in order for this to be possible the user must be authorized to do so.

When the computer has been bootstrapped, terminal number 1 is started with user number 2 logged on at this terminal. Optionally, a program is automatically started here.

4.1. User Number, Name, and Password.

Each user authorized to use the computer is assigned a user number in the range $0002 to $7FFF and a name of up to 8 characters. The user may assign himself a password of up to 8 characters.

The name and the user number are synonymous; the user number is used to identify the user within the computer, the name is the means by which the user identifies himself to the computer. Using the name here instead of the user number has the advantage that a name is usually easier to remember than a number, and also a name normaly contains redundant characters decreasing the possibility of erroneous input.

The password is a secret code word which the user assigns himself to prevent misuse of his access right to the system.

The user number determines which devices and files in the system the user may access. Further, the user number is used for accounting purposes: The system stores information about how many times and for how long the user has used the computer.

4.2. Privileged Users.

Users with user numbers less than or equal to $00FF are ´privileged´. Privileged users have certain rights in the system, which unprivileged users do not. For example, privileged users may access any device or file in the computer, they may authorize new users to use the computer, and they may abort programs running in the computer, regardless of who started the program.

4.3. The Access File.

A file called the ´Access File´ is present on some disk in the computer. In this file information about all the users that have access to the system is stored. The information in this file is:

- User number.
- User name.
- User password.
- Number of times the user has logged on to the computer.
- Total logged on time.
- Date and time of last use of the computer.
- Name of a program the user wishes to be executed immediately after log on.

This file is protected so that it may only be accessed by privileged users or the Access File Maintenace Program.

4.3.1. The Access File Maintenance Program.

This program maintains the contents of the Access File. The program allows privileged users to add or delete users from the file. Also, privileged users may obtain all information stored about the users of the system.

The program allows unprivileged users to inspect the contents of the access file except for the password information, and to alter their own password and the name of the program that should be executed immediately after log on.

>> The actual functioning of the program has not yet been determined <<

4.4. Logging On.

The following text is presented on a terminal, when nobody uses it:

```
M/2 is ready for log on. Please press the escape key:
```

This is an invitation to the user to log on to the computer, that is, to request access to a master CPU.

When the user presses the escape key, the text ´<ESC>´ will appear on the screen, and the terminal driver program (which operates the terminal controller) examines the contents of common memory to see which master CPU has the fewest users logged on. The terminal driver then tells that CPU that a log on is requested, whereupon the master CPU goes through the log on procedure as described below.

The user may, alternatively, request to be logged on to a particular master CPU, for example, if he wants to abort a program he knows to be executing on that particular CPU. The user enters the CPU number and presses the escape key, whereupon the terminal driver will request a log on on the master CPU having the number given by the user.

The operating system in the master CPU starts the execution of the log on program. This program communicates with the terminal which started the operation. The functions of the log on program are as follows:

The user is requested to enter his name and password (which is not echoed on the terminal). The program looks in the access file to see if the name and password are legal. If this is not the case, an error message is output to the terminal, and the invitation to log on appears on the terminal again.

If, however, the name and password are legal, the log on program finds the user number and the program, if any, that should be started automatically whenever the user is logged on user. After this, the user is allowed to execute programs on the master CPU.

Examples:

A log on dialog may look like this (user input shown underlined):

```
M/2 is ready for log on. Please press the escape key: <ESC>
This is CPU number 4.
M/2 log on program version dd.mm.yyyy.
Please enter your name: NAPOLEON
Please enter your password: WATERLOO  (not echoed)
User number $1542, NAPOLEON, logged on to M/2 at terminal number 15.
Date: 18.06.1815   Time: 08.12
```

or

```
M/2 is ready for log on. Please press the escape key:2<ESC>
This is CPU number 2.
M/2 log on program version dd.mm.yyyy.
Please enter your name: NAPOLEON
Please enter your password: WATERLOO  (not echoed)
User number $1542, NAPOLEON, logged on to M/2 at terminal number 15.
Date: 18.06.1815   Time: 08.12
```

4.5. Logging Off.

When the user has finished using the terminal, he must log off the computer, that is, give up his access to the computer. This is done by running the Log Off program, LOGOFF.

Running this program writes statistics about the session into the access file, and the invitation to log on appears on the terminal again.

Example:

```
>LOGOFF
M/2 log off program version dd.mm.yyyy.
User number $1542, NAPOLEON, logged off M/2 after 1 hours´ 3 minutes´ use.
Date: 18.06.1815    Time: 09.15

M/2 is ready for log on. Please press the escape key:
```

5. The Supervisor.

The Supervisor is the main part of the M/2 operating system in the master CPUs. The supervisor controlls the execution of the user programs and supplies the user with various services, such as access to input/output devices, inter-process communication, timing, etc.

The user accesses these services by issuing ´system directives´.

5.1. System Directives.

All supervisor services are invoked by issuing a system directive. A system directive is a TRAP 15 instruction. When this instruction is issued register DO must contain the ´directive number´, that is, a number that identifies the directive, and AO must contain the (logical) address of a data block containing parameters for the system directive. Upon return from the directive, DO contains a reply code, and all other registers are unchanged. The reply code -1 with the symbolic name E.IDIR means ´illegal directive number´ and is common to all directives.

To ease the programming of system directives, each directive number and reply code has a symbolic name. Further, a set of assembler macroes are supplied that set up the parameters for the directives and issue the directives.

>> The macroes have not yet been defined <<

5.2. Memory Management.

The master CPU is capable of addressing up to 16 megabyte (MB) of memory. A Memory Management Unit (MMU) is supplied to protect the programs running on a master CPU from one another, and to supply dynamic relocation of programs when they are loaded.

Addresses used in programs are ´logical´ addresses. These are not used directly to access memory, instead they go through the MMU which translates these addresses into ´physical´ addresses, the actual memory locations.

User programs may address up to 14 MB of memory, using the logical addresses $200000-$FFFFFF. This logical memory space is divided into 14 so-called segments of 1 MB each. The segments are numbered from 2 to 15. The remaining 2 MB, the would-be segments number 0 and 1, are reserved for the supervisor.

The logical addresses of these segments are:

Segment no.	Addresses
2	$200000-$2FFFFF
3	$300000-$3FFFFF
4	$400000-$4FFFFF
5	$500000-$5FFFFF
6	$600000-$6FFFFF
7	$700000-$7FFFFF
8	$800000-$8FFFFF
9	$900000-$9FFFFF
10	$A00000-$AFFFFF
11	$B00000-$BFFFFF
12	$C00000-$CFFFFF
13	$D00000-$DFFFFF
14	$E00000-$EFFFFF
15	$F00000-$FFFFFF

Each segment has the following attributes: Its length, that is, how much of the 1 MB in the segment is actually used (the length is given in 256 byte blocks), and whether the segment is a read-only segment or a read/write segment.

Let us assume, for example, that a user has a program that requires $437F4 bytes of memory for the actual program and the constants, and $223B bytes of memory for the dynamic data. Further, the program has, during its execution, requested an additional $1000 bytes of memory.

The user may use the following disposition of the logical addresses:

Segment 2 is used for the program and the constants. The logical addresses used will be $200000-$2437F3. Thus segment 2 is a read-only segment with the length $43800, giving the user access to logical addresses $200000-$2437FF.

Segment 3 is used for the dynamic data. The logical addresses used will be $300000-$30223A. Thus segment 3 is a read/write segment with the length $2400, giving the user access to logical addresses $300000-$3023FF.

The $1000 bytes requested during program execution is address by segment 4. The logical addresses used will be $400000-$400FFF. Thus segment 4 is a read/write segment with the length $1000, giving the user access to exactly the logical addresses requested.

These three segments are mapped by the MMU into physical memory. The user need not know anything about the actual location of these segments in memory. The actual mapping may, for example, be:

Logical addresses	Physical addresses
$200000-$2437FF	$2B0600-$2F3DFF
$300000-$3023FF	$2F3E00-$2F61FF
$400000-$400FFF	$4A6200-$4A71FF

Assigning segments and thus logical addresses to the program and its data is done automatically by the assembler and linker. Assigning segments to dynamically acquired memory is done explicitly by the program.

The functioning of the MMU imposes certain restrictions on the user´s dynamic allocation of memory. Because the user has only 14 segments at his disposal, and because each acquisition of additional memory requires that a new segment be used, he cannot acquire 1 kilobyte of memory 20 times, but he may acquire 20 kilobytes all at once.

6. Processes.

A process is a running program. Processes have various properties, and various operations may be performed on processes.

A process is said to ´belong´ to the user who started it, and, conversely, he is said to ´own´ the process. Generally, operations on a process can only be performed by the user owning the process or by a privileged user.

If the user owning a process is privileged, the process is itself said to be privileged. However, some program are privileged even if executed by non-privileged users. A program is given a privileged status by the ´Set Program Privilege Status´ I/O request described in section 10.6.27.

A process is identified by its name, which is a string of 8 characters, and the number of the user owning the process. Thus diffent users may have processes with identical names running simultaneously.

When a process is started, the user may either himself supply a process name, or request the supervisor to do so.

6.1. The Different Kinds of Processes.

A process is either a ´main process´, which is directly subordinate to the operating system, or it may be a ´sub-process´, which is subordinate to another process. The diffence between these two kinds of processes is mainly reflected in what happens when a process dies, that is, terminates its execution. This is described in section 6.5.

The programs for the processes may be located in three different places:

1) In a file from which it is loaded when execution starts.
2) In the memory belonging to another process.

3) ´Installed´ in memory, that is, permanently present in the master CPU memory.

A few words about installed programs are in order: Execution of installed programs can start very fast, because there is no need to load the program. One or more read/write segments for data are simply allocated and the code is executed. In this way the code for an installed program may be used in the execution of several processes. This facility may be used, for exaple, for the Pascal compiler, the Pascal interpreter, the Comal interpreter, the assembler, the various utility programs, etc. Only privileged users may install programs.

One program, RUN, see section 6.4, is always installed when M/2 is loaded. Other programs may be installed by a program that is automatically started when M/2 is loaded.

6.2. Priorities.

Processes have priorities. The priority is a number in the range 0-15, with 0 indicating the highest priority. Priority 0 is reserved for the operating system, priorities 1-7 may only be used by privileged processes, whereas priorities 8-15 are available to all users. The standard priority is 12.

The priority is used when several processes are competing for access to the CPU.

6.3. Operations on Processes.

6.3.1. Starting a Process.

Processes may be started in several ways:

1) Load and execute non-installed program.
   This creates a new main process. The program code is loaded from a file.

2) Load and spawn non-installed program.
   This creates a new sub-process, subordinate to the process issuing this request. The program code is loaded from a file.

3) Spawn in-memory program.
   This creates a new sub-process, subordinate to the process issuing this request. The program code is part of the memory of the process issuing this request.

4) Execute installed program.
   This creates a new main process. The program code is installed. This execute command may be performed on a master CPU different from the one on which the process issuing the request is executing.

5) Spawn installed program.
   This creates a new sub-process, subordinate to the process issuing this request. The program code is installed.

The operator communication part of the supervisor has a facility for method 4 only. Method 2, 3, and 5 are not possible on the operator communication level. Method 1 is brought about by having an installed program, RUN, which issues a suitable directive. The operator communication is described in section 6.4.

It is seen that the term ´spawn´ is used to designate the starting of a sub-process. The process issuing the spawn request is termed the ´parent process´, whereas the spawned process is termed the ´offspring process´.

An ´execution initiation time´ (EIT) may be specified when a process is started. This means that the actual loading and execution of the program does not start before the specified EIT. Until this time has been reached, the process is said to be ´pending´.

A ´rescheduling interval´ (RSI) may be specified when a process is started. This means that when the process dies, it will enter the pending state and will be restarted again and again at intervals specified in the RSI. The intervals will count from the start of the process execution. If the process executes longer than the specified RSI, the double, triple, or quadruple, etc., RSI will be used. The restarting of the process is terminated when the process is aborted (see section 6.3.3).

If, for example, a process is started with an EIT of 12.34 and an RSI of 1 hour, 20 minutes, it will be pending until 12.34 where execution will start. When the process dies it will enter the pending state and be restarted at 13.54 and then again at 15.14, and so on until the process is aborted.

6.3.2. Exiting a Process.

A process may terminate its execution by issuing an exit request. This involves informing its parent process, if any, that it has done so (see section 6.5), releasing all memory belonging to that process, and closing all files which the process has not closed itself.

6.3.3. Aborting a Process.

A user may abort a process, that is, force the process to exit. If the process has an RSI, this will be ignored, and the process will not be restarted after the expiration of the RSI time. Pending processes may be aborted as well, and will not start executing at the appointed time. An unprivileged user may only abort his own processes. A privileged user may abort any process.

6.3.4. Suspending a Proces.

The execution of a process may be temporarily suspended. An unprivileged user may only suspend his own processes. A privileged user may suspend any process.

The execution is resumed when the user issues a ´resume´ request or when a specified time expires.

6.3.5. Installing a Program.

A privileged user may install programs. Such programs must consist only of read-only segments and unitialized read/write segments.

When a program is installed the contents of the read-only segments are read into memory. This memory will be shared by all processes executing this program.

When an installed program is executed, the program code is located in the read-only segments already in memory. Read/write segments will be assigned to the process as required and will not be shared by the different processes executing this program.

An installed program can only be removed from memory by privileged users and only if no process is currently executing it.

6.4. Operator Communication.

The operator communication is part of the supervisor. Its job is to handle ´execute installed program´ requests from the user terminal. This is the only operation that can be performed directly from the terminal; all other requests are carried out by executing programs.

The user presses the escape key which causes the operator communication prompt

```
        >
```

to appear on the terminal, inviting the user to enter the name of an installed program.

The format of the command is:

```
        >prog parameters
or
        >prog: parametes
or
        >prog*c parameters
or
        >prog*c: parametes
```

where prog is the name of the installed program that should be executed and parameters is an optional character string that will be passed to the process.

If the parameter string is too long for one line of input, the user may specify ++ as the last two characters of the line. In this case the operator communication will output the prompt

```
        +>
```

on the next line, and the user may continue the parameter string here.

When the return key is pressed a process belonging to the user is created, executing the specified installed program. If the escape key is pressed, the command is ignored.

If the first version of the command is used, the supervisor will assign a name to the process according to the rules given in section 6.7.4. Execution will start immediately (no EIT given) and no RSI will be assumed. The process will execute in an environment specified by the global logical unit assignments given prior to this command (see section 11), and the priority will be 12.

If the second version of the command is used, the operator communication will output the prompt

```
        :>
```

whereupon the user may enter so-called subcommands specifying how the program should execute. After each subcommand the user presses the return key, whereupon another subcommand prompt appears. Entering the subcommand FIN starts the execution of the process. Pressing the escape key aborts the whole program start command. The legal subcommands are

NAME:nnnnnnnn	This subcommand specifies that the process should have the name nnnnnnnn, consisting of at most 8 characters.
LU:<aaaa>=bbbb	This subcommand specifies that the program should execute in the context given by the local assignment to the logical I/O unit <aaaa> of the I/O unit bbbb, which may contain a logical unit specification.
PRIO:nn	This subcommand specifies that the program should execute with the priority nn ranging from 1 to 15.
EIT:DATE=dd.mm.yyyy TIME=hh.mm.ss	This subcommand specifies that the program should execute at the specified date and time. The date specification may be omitted, in which case the next occurence of the specified time is assumed.
RSI:D=ddd H=hhh M=mmm S=sss	This subcommand, in which any of the parameters may be omitted, specifies the rescheduling interval. ddd, hhh, mmm, and sss are the number of days, hours, minutes, and seconds in the RSI, respectively.
FIN	This subcommand specifies that the list of

subcommands is finished and that program execution may begin.

>> The action to be taken when erroneous commands or subcommands are entered has not been determined <<

The two last forms of the ´execute installed program´ command (those with the *c after the program name) are equivalent to the two previous forms, except that the program to be executed is located on another master CPU. The c in the command is the number of the master CPU on which the installed program to be executed is located.

Examples:

If the user wants to to execute the installed program PROG with no parameters, no EIT, no RSI, no special logical unit assignments, and with a priority of 12 he may give the following command (user input shown underlined):

```
>PROG
```

If the user wants to execute the installed program PROG, pass it the parameter string ´ABC DEF GHI JKL MNO PQR STU VWX YZ´, starting execution at 12 o´clock, and having the logical units <LIST> and <ERROR> assigned the I/O units /PRINT02 and <MYTERM>, respectively, he may give the following command (user input shown underlined):

```
>PROG: ABC DEF GHI JKL MNO PQR STU ++
+>VWX YZ
:>EIT: TIME=12.00.00
:>LU:  <LIST>=/PRINT02
:>LU:  <ERROR>=<MYTERM>
:>FIN
```

6.4.1. The RUN Program.

One program is automatically installed in the system when it is loaded: The RUN program. This program is performs a ´load and execute non-installed program´ request. The format of the command is:

```
        >RUN prog parameters
```

or

```
        >RUN prog: parameters
```

In this command prog is a file specification (see section 10.1). If no disk is specified, that is, the specification does not start with an ´!´, a ´/´, or a ´<´, the logical unit, <SYSDISK>, which defaults to the physical unit /DISK01, is searched for the file. The program located in the prog file is loaded and executed. The parameter string parameters is passed to it. As specified above, the parameter string may continue on the next line if the last two characters entered are ++.

If the first version of the command is used, the supervisor will assign a name to the process according to the rules given in section 6.7.1. Execution will start immediately (no EIT given) and no RSI will be assumed. The process will execute in an environment specified by the global logical unit assignments given prior to this command (see section 11), and the priority will be 12.

If the second version of the command is used, the RUN program will output the prompt

```
        RUN>
```

whereupon the user may enter so-called subcommands specifying how the program should execute. The format of these subcommands are the same as those specified above.

Examples:

If the user wants to to load and execute the program located in the file PROG on the disk specified by the logical unit <SYSDISK> with no parameters, no EIT, no RSI, no special logical unit assignments, and a priority of 12, he may give the following command (user input shown underlined):

```
>RUN PROG
```

If the user wants to load and execute the program located in the file PROG on disk /DISK02, pass it the parameter string ´ABC DEF GHI JKL MNO PQR STU VWX YZ´, starting execution at 12 o´clock, and having the logical units LIST> and <ERROR> assigned the I/O units /PRINT02 and <MYTERM>, respectively, he may give the following command (user input shown underlined):

```
>RUN /DISK02/PROG: ABC DEF GHI JKL MNO PQR STU ++
+>VWX YZ
RUN>EIT: TIME=12.00.00
RUN>LU:  <LIST>=/PRINT02
RUN>LU:  <ERROR>=<MYTERM>
RUN>FIN
```

6.5. Process Death.

A proces may die either by committing suicide using an exit request or by being killed by another process issuing an abort request. When a process dies its death is reported to its parent process or the operating system if no parent process exists.

If a parent process dies, all of its offspring processes are automatically aborted.

When a process, A, spawns a process, B, it defines an exit semaphore. (For a discussion on semaphores, see section 7.) When B dies, a message is sent by the operating system to A´s exit semaphore. This message is two bytes long and contains the so-called completion code specified in the exit or abort request that killed B. This completion code is an integer number that can be used to inform A about the cause of the death of B. No information is sent, if the exit semaphore does not exist.

The death of a main process is reported on the terminal unless the completion code is zero. If the completion code is different from zero, the death is reported thus on the terminal:

```
USER $uuuu, nnnnnnnn, PROCESS pppppppp TERMINATED. COMPLETION CODE xxxx
```

Here uuuu is the hexadecimal number of the user who owns the process, nnnnnnnn is the name of the user, pppppppp is the name of the process, and xxxxx is the completion code in decimal.

When a process dies, it is automatically detached from all semaphores and partitions, and all its open files are closed.

6.6. System Directives.

The following system directives are used in process management:

6.6.1. Load and Execute Non-installed Program.

Symbolic name of directive number: LD.EXEC

Data block:

DC.B	name of process (8 bytes)
DC.B	file containing program,0
DC.B	EIT Date (10 characters) and time (8 characters)
DC.L	RSI in seconds
DC.W	process priority
DC.W	length of parameter string passed to process
DC.B	parameter string passed to process
DC.B	reply semaphore name (8 bytes)
DC.W	number of logical unit assignments

for each logical unit assignment this data is supplied:

DC.B	logical unit (8 bytes)
DC.B	character string which is to be assigned to the logical unit,0

This directive initiates the loading of a program and its subsequent execution as a main process. The process will belong to the same user as the process issuing the directive. The new process is not yet executing when the supervisor returns control to the process issuing the directive, the loading has merely been initiated. When the loading is complete, a message will be sent to the specified reply semaphore (see below).

The data block may contain the name which the process is going to have. If, however this field is coded as 8 blanks, the supervisor will itself create a name for the process. If the name of the file containing the program is or may be truncated to ABCDEFGH, the name of the process will be ABCDEFGH if the user does not already have a process with this name. If the user already has a process with the name ABCDEFGH, the supervisor will go through the names ABCDE.01, ABCDE.02, ABCDE.03, etc. until a name is found which is not already used by that user. If the file name contains less than 5 characters, the .01, .02, etc. will be appended to the file name. If the process name was created by the supervisor, the name will be stored in the process name field of the data block, provided that the user has read/write access to the data block.

The EIT (Execution Initiation Time) consists of a date and a time specification. No EIT is assummed if the time is specified as 8 blanks. If a time is specified and the date is given as 10 blanks, the next occurence of the time is assumed to be the EIT. The format of the date specification is ´dd.mm.yyyy´, for example, ´26.02.1982´. The format of the time specification is ´hh.mm.ss´, for example, ´23.59.59´, using a 24 hour clock.

The RSI (Rescheduling Interval) is given in seconds. No RSI is assumed if the value is specified as zero.

The process priority must lie in the range 1-15 for privileged processes, and 8-15 for unprivileged processes. The default priority of 12 is assumed, if zero is specified.

A parameter string to be passed to the started process may be specified.

A reply semaphore, belonging to the user issuing the request, must be supplied. A message will be sent to this semaphore when the loading of the program is complete. This message will contain in the first word an error code, and the following bytes may contain additional error information. The length of the message will not exceed 12 bytes.

A number of local logical unit assignments, which will pertain to the started process, may be supplied. The format of these assignment specifications is given in section 11.

6.6.2. Load and Spawn Non-installed Program.

Symbolic name of directive number: LD.SPAWN

Data block:

```
        DC.B    name of process (8 bytes)
        DC.B    file containing program,0
        DC.B    EIT Date (10 characters) and time (8 characters)
        DC.L    RSI in seconds
        DC.W    process priority
        DC.B    name of exit semaphore (8 bytes)
        DC.W    length of parameter string passed to process
        DC.B    parameter string passed to process
        DC.B    reply semaphore name (8 bytes)
        DC.W    number of logical unit assignments

        for each logical unit assignment this data is supplied:

        DC.B    logical unit (8 bytes)
```

DC.B	character string which is to be assigned to the logical unit,0

This directive initiates the loading of a program and its subsequent execution as a sub-process belonging to the process issuing the directive. When the sub-process dies, a message containing a completion code is sent to the indicated exit semaphore. This semaphore belongs to the same user as the process issuing the directive. The new process is not yet executing when the supervisor returns control to the process issuing the directive, the loading has merely been initiated. When the loading is complete, a message will be sent to the specified reply semaphore.

The contents of the data block is similar to that specified in section 6.6.1, except that the name of the exit semaphore must be included.

6.6.3. Spawn In-memory Program.

Symbolic name of directive number: IM.SPAWN

Data block:

DC.B	name of process (8 bytes)
DC.L	logical address of first instruction in program
DC.B	EIT Date (10 characters) and time (8 characters)
DC.L	RSI in seconds
DC.W	process priority
DC.W	length of parameter string passed to process
DC.B	parameter string passed to process
DC.W	number of logical unit assignments

for each logical unit assignment this data is supplied:

DC.B	logical unit (8 bytes)
DC.B	character string which is to be assigned to the logical unit,0

This directive creates a sub-process subordinate to the calling process. The code for the offspring process must be part of the parent process´ memory, and execution starts at the indicated logical address. When the sub-process dies, a message containing a completion code is sent to the indicated exit semaphore. This semaphore belongs to the same user as the process issuing the directive.

The contents of the data block is similar to that specified in section 6.6.1, except that the name of the exit semaphore must be included, and instead of a file name, a logical starting address is given. If the supervisor is required to create a process name, the basis for the creation is the name of the parent process.

The offspring process has access to the same logical addresses as the parent process. When the parent process attaches to a partition (see section 9), the offspring process will automatically have access to it, and, conversely, if the offspring process attaches to a partition, the parent process will automatically have access to it.

6.6.4. Execute Installed Program.

Symbolic name of directive number: IN.EXEC

Data block:

DC.B	name of process (8 bytes)
DC.B	name of installed program (8 bytes)
DC.B	EIT Date (10 characters) and time (8 characters)
DC.L	RSI in seconds
DC.W	process priority
DC.W	CPU number
DC.W	length of parameter string passed to process
DC.B	parameter string passed to process
DC.W	number of logical unit assignments

for each logical unit assignment this data is supplied:

DC.B	logical unit (8 bytes)
DC.B	character string which is to be assigned to the logical unit,0

This directive creates a new main process executing the indicated installed program. The contents of the data block is similar to that specified in section 6.6.1, except that the name of the installed program is given instead of the name of the file containing the program, and a new parameter ´CPU number´ is supplied.

The process priority specified when the program was installed is used if a priority of zero is specified in the above data block.

The data block may contain the name which the process is going to have. If, however this field is coded as 8 blanks, the supervisor will itself create a name for the process. If the name of the installed program is ABCDEFGH, the name of the process will be ABCDEFGH if the user does not already have a process with this name. If the user already has a process with the name ABCDEFGH, the supervisor will go through the names ABCDE.01, ABCDE.02, ABCDE.03, etc. until a name is found which is not already used by that user. If the name of the installed program contains less than 5 characters, the .01, .02, etc. will be appended to the program name. If the process name was created by the supervisor, the name will be stored in the process name field of the data block, provided that the user has read/write access to the data block.

The program to be executed is assumed to reside on the same master CPU as the process issuing the directive if the CPU number is specified as zero. Otherwise, the CPU number specifies the number of the master CPU on which the program resides and should execute.

6.6.5. Spawn Installed Program.

Symbolic name of directive number: IN.SPAWN

Data block:

DC.B	name of process (8 bytes)
DC.B	name of installed program (8 bytes)
DC.B	EIT Date (10 characters) and time (8 characters)
DC.L	RSI in seconds
DC.W	process priority
DC.B	name of exit semaphore (8 bytes)
DC.W	length of parameter string passed to process
DC.B	parameter string passed to process
DC.W	number of logical unit assignments

for each logical unit assignment this data is supplied:

DC.B	logical unit (8 bytes)
DC.B	character string which is to be assigned to the logical unit,0

This directive creates a sub-process belonging to the calling process. When the sub-process dies, a message containing a completion code is sent to the indicated exit semaphore. This semaphore belongs to the same user as the process issuing the directive.

The contents of the data block is similar to that specified in section 6.6.1, except that the name of the exit semaphore must be included, and the name of the installed program is given instead of the name of the file containing the program.

The process priority specified when the program was installed is used if a priority of zero is specified in the above data block.

6.6.6. Install Program.

Symbolic name of directive number: INS.PROG

Data block:

DC.B	name of program (8 bytes)
DC.B	file containing program,0
DC.W	default process priority
DC.W	number of logical unit assignments

for each logical unit assignment this data is supplied:

DC.B	logical unit (8 bytes)
DC.B	character string which is to be assigned to the logical unit,0

This directive installs the program contained in the specified file.

The data block must contain the name which the program is going to have.

The default process priority must lie in the range 1-15. A default priority of 12 is assumed, if zero is specified.

A number of local logical unit assignments, which will pertain to a process executing this installed program, may be supplied. The format of these assignment specifications is given in section 11.

This directive may only be issued by privileged processes.

6.6.7. Remove Installed Program.

Symbolic name of directive number: REM.PROG

Data block:

```
        DC.B     name of program (8 bytes)
```

This directive removes the specified installed program from memory. No processes may currently be executing the specified program.

This directive may only be issued by privileged processes.

6.6.8. Exit.

Symbolic name of directive number: EXIT.PRC

Data block:

```
        DC.W     completion code
```

This directive causes the issuing process to die. The completion code is reported to the exit semaphore, if any, or, otherwise, to the operating system.

6.6.9. Abort Process.

Symbolic name of directive number: ABO.PRC

Data block:

```
        DC.B     name of process (8 bytes)
        DC.W     user number
        DC.W     completion code
```

This directive aborts the process with the indicated name belonging to the indicated user. Unprivileged users may only abort processes belonging to themselves. The user number of the issuing process is assumed, if the user number is specified as $FFFF. The comletion code is reported to the exit semaphore, if any, or, otherwise, to the operating system.

6.6.10. Suspend Process.

Symbolic name of directive number: SUSP.PRC

Data block:

DC.B	name of process (8 bytes)
DC.W	user number
DC.L	time in centiseconds

This directive suspends the execution of the process with the indicated name belonging to the indicated user. The process issuing the call is itself suspended if the name is specified as 8 blanks. The user number of the issuing process is assumed, if the user number is specified as $FFFF. Unprivileged users may only suspend processes belonging to themselves.

Execution is suspended indefinately if the time is specified as zero.

The execution of the process is resumed upon the issuing (by another process) of a resume directive or the expiration of the specified time, whichever comes first.

If a process has suspended itself, it will when execution is resumed in register AO find the value zero, if execution was resumed because of an expired time. The value E.RESUME will be found in AO if execution was resumed becaus of a resume directive issued by another process.

6.6.11. Resume Process.

Symbolic name of directive number: RSUM.PRC

Data block:

DC.B	name of process (8 bytes)
DC.W	user number

This directive resumes the execution of the suspended process with the indicated name belonging to the indicated user. The user number of the issuing process is assumed, if the user number is specified as $FFFF. Unprivileged users may only resume processes belonging to themselves.

6.6.12. Get Process Status.

Symbolic name of directive number: PRC.STAT

Data block:

DC.B	name of process (8 bytes)
DC.W	user number
DC.L	logical address of a word-aligned ?-byte memory location to which the issuing process has read/write access

This directive fetches the status information for the specified process belonging to the specified user. The issuing process itself is assumed if the name is specified as 8 blanks. The user number of the issuing process is assumed, if the user number is specified as $FFFF. The information is stored in the memory location, whose logical address is given in the data block.

>> The format of the status information has not yet been determined <<

6.6.13. Get Parameter String.

Symbolic name of directive number: GET.PARM

Data block:

DC.W	maximum allowable length in bytes of parameter string
DC.L	logical address of a word-aligned data area of at least the length specified as the first parameter of this data block plus two. The process issuing the directive must have read/write access to this data area. The layout of the area is:

DS.W	1	(here the actual length of the parameter string will be stored)
DS.B	n	(here the parameter string will be stored)

This directive fetches the parameter string passed to the issuing process when it was started. If the length of the parameter string exceeds the specified maximum length, the actual length will still be stored as indicated above, but the parameter string stored will be truncated to the specified maximum length.

This directive may be issued several times by the same process.

7. Semaphores.

Semaphores are used for message exchange within the system. Before a process may use a semaphore the process must be ´attached´ to that semaphore. A semaphore is automatically deleted when no process is attached to it and no message is waiting on it, except for the so-called system semaphores which must be deleted by an explicit request. Privileged processes may communicate with any semaphore in the system. Unprivileged processes may communicate with semaphores belonging to the same user as the processes themselves, and with system semaphores (belonging to user number zero).

7.1. Operations on Semaphores.

7.1.1. Creating a Semaphore.

A user process may create a semaphore. The semaphore is given a name and the process is automatically attached to it. The semaphore is said to ´belong´ to the user issuing the request.

7.1.2. Attaching to a Semaphore.

Before a process can send a message to or receive a message from a semaphore, the process must be attached to that semaphore. System directives exist for attaching to and detaching from semaphores.

7.1.3. Message Exchange.

A process may send a message to or receive a message from a semaphore. A queue of messages or processes may exist on a semaphore.

Sending a message:

- If no process is waiting on the semaphore, the message is placed as the last element in the queue on the semaphore.
- If one or more processes are waiting in the semaphore queue, the message is given to the first of these, and the execution of that process is resumed.

Receiving a message:

- If no message is available in the semaphore queue, the process is placed as the last element in the queue and its execution is suspended.
- If one or more messages are available in the semaphore queue, the first of these is given to the process and its execution continues.

7.2. System Directives.

The following system directives exist:

7.2.1. Create Semaphore.

Symbolic name of directive number: CREA.SEM

Data block:

```
        DC.B    semaphore name (8 bytes)
```

This directive creates a semaphore with the specified name. This semaphore will belong to the user, to whom the process issuing the directive belongs. The process issuing the directive is automatically attached to the semaphore.

7.2.2. Create System Semaphore.

Symbolic name of directive number: CR.SYSEM

Data block:

DC.B	semaphore name (8 bytes)

This directive creates a semaphore with the specified name. This semaphore will belong to user number zero (the system). The process issuing the directive is automatically attached to the semaphore. This directive may only be issued by privileged processes.

7.2.3. Delete System Semaphore.

Symbolic name of directive number: DE.SYSEM

Data block:

DC.B	semaphore name (8 bytes)

This directive deletes the semaphore with the specified name. This semaphore must belong to the system (user number zero), and no processes must be attached to it, nor may any messages be available on it. This directive may only be issued by privileged processes.

7.2.4. Attach Semaphore.

Symbolic name of directive number: ATT.SEM

Data block:

DC.B	semaphore name (8 bytes)
DC.W	user number

This directive attaches the process issuing the call to the specified semaphore belonging to the specified user. Unprivileged processes may only attach to semaphores belonging to user number zero (the system) or to the user to whom the process belongs. The user number of the issuing process is assumed, if the user number is specified as $FFFF.

7.2.5. Detach Semaphore.

Symbolic name of directive number: DET.SEM

Data block:

DC.B	semaphore name (8 bytes)
DC.W	user number

This directive detaches the process issuing the call from the specified semaphore belonging to the specified user. The process must be attached to the semaphore when the directive is issued. The user number of the issuing process is assumed, if the user number is specified as $FFFF.

7.2.6. Send Message to Semaphore.

Symbolic name of directive number: SEND.SEM

Data block:

DC.B	semaphore name (8 bytes)
DC.W	user number
DC.B	reply semaphore name (8 bytes)
DC.W	message length in bytes
DC.B	actual message

This directive sends a message to the specified semaphore belonging to the specified user. The user number of the issuing process is assumed, if the user number is specified as $FFFF. The process issuing the directive must be attached to the semaphore before the directive is issued. The reply semaphore is assumed to belong to the user to whom the process issuing the directive belongs. No reply semaphore is assumed if the reply semaphore name is specified as 8 blanks.

7.2.7. Receive Message from Semaphore

Symbolic name of directive number: REC.SEM

Data block:

DC.B	semaphore name (8 bytes)
DC.W	user number
DC.W	maximum allowable message length, n
DC.L	the logical address of a word-aligned data area to which the process has read/write access. The layout of this area must be as follows:

DS.B	8	(here the name of the reply semaphore will be stored)
DS.W	1	(here the user number of the reply semaphore will be stored)
DS.W	1	(here the length of the message will be stored)
DS.B	n	(here the message will be stored)

This directive waits for and receives a message from the specified semaphore belonging to the specified user. The user number of the issuing process is assumed, if the user number is specified as $FFFF. The process issuing the directive must be attached to the semaphore before the directive is issued. If the length of the message is greater than the maximum allowable message length, nothing will be received, but a directive error will be reported.

7.2.8. Remove Message from Semaphore.

Symbolic name of directive number: REM.MES

Data block:

DC.B	semaphore name (8 bytes)
DC.W	user number

This directive removes the first message, if any, from the specified semaphore belonging to the specified user. The user number of the issuing process is assumed, if the user number is specified as $FFFF.

This directive is primarily intended to remove a message which is too long to be received.

7.2.9. Get Semaphore Status.

Symbolic name of directive number: SEM.STAT

Data block:

DC.B	semaphore name (8 bytes)
DC.W	user number
DC.L	the logical address of a word-aligned ?-byte data area to which the issuing process has read/write access.

This directive fetches the status information for the specified semaphore belonging to the specified user. The user number of the issuing process is assumed, if the user number is specified as $FFFF. The information is stored in the data area whose logical address is given in the data block.

>> The format of the status information has not yet been determined <<

8. Symbolic Resources.

Symbolic resources are character strings that represent various physical resources, such as files, in the computer. By reserving a symbolic resource, a process indicates its desire to use the physical resource associated with it. There is, however, no actual connection between the symbolic resource and the physical one. The connection is established only by the mutual agreement of the processes that they will follow a certain convention.

Symbolic resources are global to all master CPUs in the computer and may thus be used, for example, by file systems.

Only privileged processes may operate on symbolic resources.

8.1. Operations on Symbolic Resources.

Symbolic resources may be reserved and released. The reservation may be either exclusive or non-exclusive. Several non-exclusive reservations are allowed simultaneously on the same resource, but if exclusive reservation is desired, that must be the only reservation of the resource.

This means that if a resource has been exclusively reserved, all other reservation requests will be rejected. If a resource has been non-exclusively reserved, exclusive reservation requests will be rejected, bur non-exlucisve requests will be allowed.

There is no requirement that a resource be released by the process that reserved it.

8.2. System Directives.

The following system directives exist.

8.2.1. Reserve Resource.

Symbolic name of directive number: RES.RES

Data block:

DC.B	reservation kind (0 for non-exclusive, 1 for exclusive reservation)
DC.B	resource name,0

This directive reserves the symbolic resource with the specified name exclusively or non-exclusively, as specified. The reservation is rejected if the resource has already been reserved in a manner that disallows the desired reservation.

This directive may only be issued by privileged processes.

8.2.2. Release Resource.

Symbolic name of directive number: REL.RES

Data block:

DC.B	resource name,0

This directive releases the symbolic resource with the specified name.

This directive may only be issued by privileged processes.

8.2.3. Get Resource Information.

Symbolic name of directive number: GET.RES

Data block:

DC.W	maximum allowable length in bytes of resource name.
DC.B	resource name,0
DC.L	logical address of a word-aligned data area to which the process issuing the directive has read/write access. The length of the data area must be at least the length specified as the first parameter of this data block plus 4. The layout of the data area is:

DS.W	1	(here 0 will be stored for non-exclusive reservations, 1 for exclusive reservations)
DS.W	1	(here the number of times the resource has been reserved will be stored. This number is always 1 for exclusive reservations)
DS.B	n	(here the name of the resource will be stored followed by a 0-byte)

This directive gets the name and reservation information of the reserved symbolic resource that follows the specified resource name in an alphabetical order.

9. Memory.

As described in section 5.2 the user has access to up to 14 megabyte of memory. Normally, one or more read-only segments and one or more read/write segments are allocated to the user process. The user may desire to allocate additional memory during program execution, or the user may wish to access memory allocated by another process. This section describes how this is done.

9.1. Partitions.

Data areas allocated during program execution are termed ´partitions´. A process may create a partition, and, optionally, allow other processes to access this partition.

One special use of partitions is for resident subroutine libraries. Partitions available to all users may be created, and these partitions may, for example, contain often-used subroutines. The user programs may access these subroutines simply by ´attaching´ (see below) to the appropriate partition.

Before a process may use a partition the process must be ´attached´ to that partition. A partition is deleted when no process is attached to it, except for the so-called system partitions, which must be deleted by an explicit request.

Privileged processes may attach to any partition in the system. Unprivileged processes may attach to partitions belonging to the same user as the processes themselves, and to system partitions (belonging to user number zero).

Processes attaching to a partition will map certain logical addresses onto the physical addresses of the partition, using one segment per partition. Thus the maximum of 14 segments accessible to each process sets a limit to the number of partitions to which a process can be attached at any given time.

9.2. Operations on Partitions.

9.2.1. Creating a Partition.

A user process may create a partition. The partition is given a name and the process is automatically attached to it. A certain memory segment (certain logical addresses) are mapped to the physical addresses of the partition. The partition is said to belong to the user issuing the request. When a process creates a partition, it specifies if other processes may access the partition, and if so, if they are allowed read/write access or only read-only access to the partition.

9.2.2. Attaching to a Partition.

Before a process can access a partition, the process must be attached to that partition. When a process is attached to a partition, a certain memory segment (certain logical addresses) are mapped to the physical addresses of the partition.

9.3. System Directives.

The following system directives exist:

9.3.1. Create Partition.

Symbolic name of directive number: CREA.PAR

Data block:

DC.B	partition name (8 bytes)
DC.L	partition length in bytes (at most $100000)

DC.L	partition logical address (only bits 20-23 specifying the segment number are used)
DC.B	access (0 for none, 1 for read-only, 2 for read/write)

This directive creates a partition with the specified name. This partition will belong to the user, to whom the process issuing the directive belongs. The process issuing the directive is automatically attached to the partition. The process issuing the directive is allowed read/write access to the partition. Other processes may request read/write or read-only access depending on the value of the access byte in the above data block. The length of the partition will be the smallest multiple of 256 bytes greater than or equal to the requested length. The partition will be mapped to the logical addresses of the issuing process as specified in the data block.

9.3.2. Create System Partition.

Symbolic name of directive number: CR.SYPAR

Data block:

DC.B	partition name (8 bytes)
DC.L	partition length in bytes (at most $100000)
DC.L	partition logical address (only bits 20-23 specifying the segment number are used)
DC.B	access (1 for read-only, 2 for read/write)

This directive creates a partition with the specified name. This partition will belong to the system (user number zero). The process issuing the directive is automatically attached to the partition. The process issuing the directive is allowed read/write access to the partition. Other processes may request read/write or read-only access depending on the value of the access byte in the above data block. The length of the partition will be the smallest multiple of 256 bytes greater than or equal to the requested length. The partition will be mapped to the logical addresses of the issuing process as specified in the data block. This directive may only be issued by privileged processes.

9.3.2. Delete System Partition.

Symbolic name of directive number: DE.SYPAR

Data block:

DC.B	partition name (8 bytes)

This directive deletes the partition with the specified name. This partition must belong to the system (user number zero), and no processes must be attached to it. This directive may only be issued by privileged processes.

9.3.4. Attach Partition.

Symbolic name of directive number: ATT.PAR

Data block:

DC.B	partition name (8 bytes)
DC.W	user number
DC.L	partition logical address (only bits 20-23 specifying the segment number are used)
DC.B	access (0 for read-only, 1 for read/write)

This directive attaches the process issuing the call to the specified partition belonging to the specified user. Unprivileged processes may only attach to partitions belonging to user number zero (the system) or to the user to whom the process belongs. The user number of the issuing process is assumed, if the user number is specified as $FFFF. The partition will be mapped to the specified logical address. The access byte in the above data block specifies what kind of access the user requests to the partition.

9.3.5. Detach Partition.

Symbolic name of directive number: DET.PAR

Data block:

DC.B	partition name (8 bytes)
DC.W	user number

This directive detaches the process issuing the call from the specified partition belonging to the specified user. The user number of the issuing process is assumed, if the user number is specified as $FFFF. The process must be attached to the partition when the directive is issued.

9.3.6. Get Partition Status.

Symbolic name of directive number: PAR.STAT

Data block:

DC.B	partition name (8 bytes)
DC.W	user number
DC.L	the logical address of a word-aligned ?-byte data area to which the issuing process has read/write access.

This directive fetches the status information for the specified partition belonging to the specified user. The user number of the issuing process is assumed, if the user number is specified as $FFFF. The information is stored in the data area whose logical address is given in the data block.

>> The format of the status information has not yet been determined <<

9.3.7. Get Memory Information.

Symbolic name of directive number: MEM.INFO

Data block:

DC.L	the logical address of a word-aligned ?-byte data area to which the issuing process has read/write access.

This directive fetches information about the memory usage in the computer. The information is stored in the data area whose logical address is given in the data block.

>> The format of the information has not yet been determined <<

10. I/O.

All I/O requests are handlede by a single system directive. This directive specifies what is going to happen on which I/O unit.

An I/O unit may, for example, be a terminal, a printer, a disk, or a file on a disk. All these types of units are handled in a uniform manner, the only difference is the kind of requests allowed on the unit. For example, a read operation is (as seen from the user program´s point of view) identical on a terminal and a disk file. On the other hand, a ´delete file´ operation is not allowed on a terminal, whereas a ´define attention semaphore´ operation is not allowed on a printer or a disk file.

I/O to a disk may be performed on the sector level using the commands ´input sectors´ and ´output sectors´. However, it will often be desired to impose a file system on a disk. A file system is a process. When a disk has a file system imposed on it, all I/O requests on the file level will be redirected from the supervisor to the file system process, which translates the file operations into ´input sectors´ and ´output sectors´ requests. Various disks in the computer may have different file system imposed on them.

10.1. Unit Specification.

An I/O unit specification is a character string consisting of:

- A computer specification, which identifies the computer within the MikNet on which the unit resides. This specification consist of up to 8 ASCII characters.
- An I/O device specification, which identifies the physical device on which the I/O operation is to take place. This may, for example, be a disk, a terminal, or a printer. This specification consists of up to 8 ASCII characters.
- A file name specification, which identifies the disk file, if any, on which the I/O operation is to take place. The format

of this specification depends on the file system imposed on the disk.

The computer specification starts with an exclamation mark, and the specification of each of the three items above is seperated by a ´/´ character. For example, the file ALPHA.K on disk DISK01 residing on computer COMP1 is specified as ´!COMP1/DISK01/ALPHA.K´.

The computer specification may be omitted, in which case the computer on which the process in question runs is used. For example, the file ALPHA.K on disk DISK01 residing on the same computer as the program accessing the file is specified as ´/DISK01/ALPHA.K´.

The I/O device specification may be omitted with certain I/O operations. For example, an ´open file´ request will search all disks for the specified file if the disk specification is omitted. Thus the specification ´//ALPHA.K´ in this case denotes: Search all disks on ´this´ computer for the file ALPHA.K.

The file name specification may or may not be present depending on the I/O operation performed and the I/O device used. Some I/O devices do not have file systems imposed on them. For example, the terminal TERM01 on ´this´ computer is specified as ´/TERM01´.

10.2. I/O System Directive.

There is one system directive to handle all I/O requests.

Symbolic name or directive number: IN.OUT

Data block:

DC.W	request identification number
DC.W	request code
DC.B	reply semaphore name (8 bytes)
DC.W	parameter block length in bytes
DC.x	parameter block

This directive requests the supervisor, possibly using a file system, to perform a certain I/O operation as specified in the request code. When the I/O operation is terminated a message will be sent to the specified reply semaphore, which is assumed to belong to the user to whom the process issuing the directive belongs. No reply message will be sent if the reply semaphore name is specified as 8 blanks. The length of the message sent to the reply semaphore depends on the request. The message will have the following format:

```
        DC.W    request identification number from data block
        DC.W    request code from data block
        DC.W    error code
          :
          :     additional request-dependent information
          :
```

The request identification number is not used by the supervisor or the file system, it merely aids the user in identifying which of the possibly many issued I/O requests that has been completed.

The contents of the parameter block depends on the I/O request. The parameter block consists of one or more parameters in the following format:

```
        DC.W    parameter type
        DC.x    parameter
```

the length of the parameter depends on the parameter type.

The parameters may be given in any order. If a parameter, which is not required by a specific I/O request, is given, it is ignored. If a parameter is omitted, a default value may apply.

The actual I/O requests, the parameters used, and the format of the reply messages are described in section 10.6.

10.3. I/O Unit Protection.

A protection scheme is devised to prevent any user from accessing any I/O device.

The supervisor stores with each I/O device an ´Access Right Descriptor´. This ARD consists of 4 bytes describing which users have access to the device.

Byte 0-1:	Access user number.
Byte 2:	Number of significant bits in byte 0-1 minus 1 when read/write access is being considered. The contents of this byte are called the read/write access level.
Byte 3:	Number of significant bits in byte 0-1 minus 1 when read only access is being considered. The contents of this byte are called the read only access level.

Let the indicated access user number be called A, let the number of the user wanting access be called U, and let the (read/write or read only) access level be L. The user has (read/write or read only) access if and only if the L+1 most significant bits of A are equal to the L+1 most significant bits of U.

Note: Privileged users have access to any device regardless of their user number.

Example:

Assume that the ARD for a particular I/O device looks like this:

Byte 0-1:	Access user number: $5D25
Byte 2:	Read/write access level: 11, that is, the first 12 bits of $5D25 are significant when read/write access is being considered.
Byte 3:	Read only access level: 7, that is, the first 8 bits of $5D25 are significant when read only access is being considered.

All users whose user numbers start with $5D2 have read/write access to the device. All users whose user numbers start with $5D have read only access to the device.

It is seen that setting the access level to 0 amounts to granting all users access to the device, because the first bit of any user number is zero. Setting the access level to 15 (decimal) amounts to preventing all users but the user having the specified access user number from accessing the device.

The implementation of file protection is the duty of the file system.

The following system directives are available to handle access rights to I/O devices:

10.3.1. Change Access Right Descriptor.

Symbolic name of directive number: CH.ARD

Data block:

```
        DC.B    I/O device specification,0
        DC.W    access user number
        DC.B    read/write access level
        DC.B    read only access level
```

This directive changes the Access Right Descriptor of the specified I/O device. The access user number may be coded as $FFFF, in which case the user number of the issuing process is assumed.

This directive may only be issued by processes belonging to users having read/write access to the specified I/O device (including privileged users).

10.3.2. Get Access Right Descriptor.

Symbolic name of directive number: GET.ARD

Data block:

DC.B	I/O device specification,0
DC.L	logical address of a word-aligned 4 byte data area to which the process issuing the directive has read/write access.

This directive stores the Access Right Descriptor of the specified I/O device in the specified memory location.

10.4. File Systems.

A file system may be imposed on a disk. A file system is a privileged process converting file system requests to ´input sectors´ and ´output sectors´ requests on a disk. The process belongs to user number 1.

A file system must conform to the supervisor/file system interface specified in section 10.4.2.

The MIKFILE file system is automatically imposed on a disk in the computer when the operating system is loaded.

10.4.1. System Directives.

A file system may be imposed on and removed from a disk using the following directives:

10.4.1.1. Impose File System.

Symbolic name of directive number: IMP.FSYS

Data block:

DC.B	disk identification (8 bytes)
DC.B	file system process name (8 bytes)
DC.B	file containing file system,0
DC.W	process priority

This directive imposes the specified file system on the specified disk. No other file system must be imposed on the file when this directive is issued.

If the file system is not imposed on any disk on the computer, when the directive is issued, it is loaded from the specified file and executed using the specified process priority. It will be assigned the user number 1, regardless of the user number of the process issuing the directive.

If the file system is already imposed on a disk on the computer, that is, the file system process name is already in use, it is not reloaded, and the specified file name and priority are ignored.

Note:
The impose file system operation takes place only on the master CPU on which the process issuing the directive is executing.

This directive may only be issued by privileged processes.

10.4.1.2. Remove File System.

Symbolic name of directive number: REM.FSYS

Data block:

DC.B	disk identification (8 bytes)

This directive removes a file system imposed on the specified disk. All files on the disk must be closed before this directive is issued.

If the file system is not imposed on any other disk in the computer, it is removed from the memory of all master CPUs.

This directive may only be issued by privileged processes.

10.4.1.3. Get File System.

Symbolic name of directive number: GET.FSYS

Data block:

DC.B	I/O device specification,0
DC.L	logical address of a word-aligned 8 byte data area to which the process issuing the directive has read/write access.

This directive stores the name of the file system process imposed on the specified I/O device, which must be a disk, in the specified data area.

10.4.2. File System Requirements.

A file system must meet the following requirements:

It must create a semaphore, whose name must be the name of the file system process with the first character replaced by a ´$´ character.

On this semaphore it will receive file operation request messages from the supervisor. The format of these request messages are as follows:

DC.W	request identification number
DC.W	request code
DC.B	reply semaphore name (8 bytes)
DC.W	user number
DC.W	parameter block length in bytes
DC.x	parameter block

The file system must convert this request to ´input sectors´ and ´output sectors´ requests to the disk and issue the appropriate system directives.

When the operations are completed, the file system must send a reply message to the specified reply semaphore belonging to the user issuing the directive. This reply message must be in the format specified in section 10.2.

The parameter block is identical to the parameter block specified by the user in the IN.OUT system directive.

The physical layout of the disk may be determined by the ´Get Hardware Configuration´ system directive described in section 13.4.

When the file system opens a file it must inform the supervisor that the process has yet another file open. When a file is closed, the file sytem must inform the supervisor about that. The supervisor uses this information 1) to determine if a Remove File System directive is legal, 2) to issue a Close All Files request to the file sytem when a process dies (see below).

The system directive used to inform the supervisor that a file has been opened or closed is the following:

Symbolic name of directive number: FILE.USE

Data block:

DC.W	0 if a file has been closed, 1 if a file has been opened.
DC.B	name of process accessing file (8 bytes).
DC.W	user number of process accessing file.
DC.B	disk name (8 bytes).

The file system must support an I/O request, CLO.ALL, Close All Files. This request is issued by the supervisor when a process dies, and all its open files should be closed. The format of this request message is:

DC.W	0	(request identification)
DC.W	CLO.ALL	(request code)
DC.B	reply semahore (8 bytes)	
DC.W	user number of dying process	
DC.W	PROC.NAM	
DC.B	name of dying process (8 bytes)	

When this request is received the file system should close all open files belonging to the specified process.

A file system <u>should</u> meet the following requirements:

The file system should be able to accept a new request from a user process before a previous one has been terminated. Using a practical request identification numbering makes it possible to use the same semaphore to receive user requests and replies to the file system´s own requests.

The file system should impose a file access rights system, analogous to that used on I/O devices, on the files it handles.

10.5. I/O Units.

The following I/O devices are present on a computer:

TERM01, TERM02, TERM03, etc.	- the terminals.
PRINT01, PRINT02, PRINT03, etc.	- the printers.
DISK01, DISK02, DISK03, etc.	- the disks.
NULL	- a null (dummy) device.

The NULL device simply ignores all data sent to it, and returns an end-of-file condition when an input request is issued. Any file specification is legal on the NULL device.

10.6. I/O Operations.

This section describes the I/O operations implemented on the terminals, the printers, the null device, disks without file systems, and disks with the MIKFILE file system.

The available request codes are:

Symbolic name of request code	Request name	Available on
CR.FILE	Create File	MIKFILE
CR.EXT	Create Extent	MIKFILE
OPN.UNIT	Open I/O Unit	all I/O units
CLO.UNIT	Close I/O Unit	all I/O units
POS.FILE	Position File	MIKFILE
IN.FREC	Input Fixed Length Record	MIKFILE
OUT.FREC	Output Fixed Length Record	MIKFILE
IN.VREC	Input Variable Length Record	MIKFILE, terminals,
OUT.VREC	Output Variable Length Record	MIKFILE, terminals,
BIN.VREC	Backwards Inp.Var.Len. Record	MIKFILE
BOU.VREC	Backwards Outp.Var.Len. Record	MIKFILE
REN.FILE	Rename File	MIKFILE
DEL.FILE	Delete File	MIKFILE
EDIT.BUF	Update Character String	MIKFILE, terminals
DEF.ATTN	Define Attention Semaphore	terminals
REV.ATTN	Revert Attention Semaphore	terminals
ENTER.DI	Enter Direct Input Mode	terminals
EXIT.DI	Exit Direct Input Mode	terminals
DEF.FKEY	Define Function Key Area	terminals
CAT.DISK	Get Catalog Item	MIKFILE
IN.SECT	Input Sectors	disks
OUT.SECT	Output Sectors	disks
SET.PRIV	Set Program Privilege Status	MIKFILE
GET.LOAD	Get Load Module Information	MIKFILE
LOAD.MOD	Load Load Module	MIKFILE
CLO.ALL	Close All Files	MIKFILE

>> The functioning of these requests on the NULL device has not yet been determined <<

The parameters used for some or all of the above requests are:

Symbolic name of parameter type	Parameter name	Parameter format
UNIT.NAM	I/O Unit Name	Byte string terminated by 0-byte
UNIT.NA2	Secord I/O Unit Name	Byte string terminated by 0-byte
BUF.ADDR	Buffer Address	Long word
BUF.LENG	Buffer Length	Long word
STRT.SEC	Starting Sector	Long word
OUTP.CNT	Output Controls	Word
ECHO.LF	Echo Line Feed	Word
NOT.MOD	Do Not Modify	Word
CUR.OFFS	Cursor Offset	Word
ATTN.SEM	Attention Semaphore	8 bytes
DI.SEM	Direct Input Semaph.	8 bytes
DCB.SIZE	DCB Size	Word
DCB.ID	DCB ID Number	Word
FILE.SIZ	File Size	Long word
REC.LNGT	Record Length	Long word
OPEN.TYP	Open Type	Word
RES.TYP	Reservation Type	Word
REC.NO	Record Number	Long word
AC.RIGHT	Access Right	4 bytes
FILE.ID	File Identification	Long word
FILE.IDA	File Id. Address	Long word
PROC.NAM	Process Name	8 bytes
PRIV.VAL	Privilege Value	Word
SEG.ADDR	Segment Addresses	16 long words

10.6.1. Create File.

Symbolic name of request code: CR.FILE

Available on: MIKFILE files.

Parameters:

UNIT.NAM	- Name of file to be created, including disk specification.
DCB.SIZE	- Size in sectors of Data Control Block used by MIKFILE file system. Default: 4.
DCB.ID	- DCB identification number to be used with file.
FILE.SIZ	- Size in sectors of primary file. Default: 10.
REC.LNGT	- Record length. Irrelevant for variable length record files. Default: 255.
AC.RIGHT	- Access Right Descriptor. Default: user number, 15, 15.
RES.TYP	- Reserve the file exclusively or non-exclusively. The parameter must have the value 0 for non-exclusive reservation, 1 for exclusive reservation. Default: 1.

This request creates a file with the specified name on the specified disk. The file will be opened for read/write access.

10.6.2. Create Extent.

Symbolic name of request code: CR.FILE

Available on: MIKFILE files.

Parameters:

DCB.ID	- DCB identification for the file for which an extent is to be created.

This request creates an extent for the specified file. The file must be open for read/write access.

10.6.3. Open I/O Unit.

Symbolic name of request code: OPN.UNIT

Available on: All I/O units.

Parameters:

UNIT.NAM	- Name of I/O unit to be opened. If the unit is a file, a disk need not be specified, in which case all disks are searched for a file with the specified name.
DCB.SIZE	- Size in sectors of Data Control Block used by MIKFILE file system. Relevant only for MIKFILE files. Default: 4.
DCB.ID	- DCB identification number to be used with unit.
REC.LNGT	- Record length. Relevant only for MIKFILE files. Default: The value specified when the file was created. This value is also used if a record length of zero is specified.
OPEN.TYP	- Open for read-only or read/write. The parameter must have the value 0 for read-only, 1 for read/-write reservation. Default: 0.
RES.TYP	- Reserve the unit exclusively or non-exclusively. The parameter must have the value 0 for non-exclusive reservation, 1 for exclusive reservation. Default: The value of OPEN.TYP.

This request opens the specified I/O unit and assigns the specified DCB identification number to it. For printers and disks without file systems this request reserves the I/O unit for the process issuing the request.

10.6.4. Close I/O Unit.

Symbolic name of request code: CLO.UNIT

Available on: All I/O units.

Parameters:

DCB.ID	- DCB identification number for the unit to be closed.

This I/O request closes the I/O unit. For printers and disks without file systems, this request releases the I/O unit reservation. After this request the DCB identification number may be used for another file.

10.6.5. Position File.

Symbolic name of request code: POS.FILE

Available on: MIKFILE files.

Parameters:

DCB.ID	- DCB identification number for the file to be positioned.
REC.NO	- Number of record to which the file is to be positioned. Records are numbered 1, 2, etc. Default: 1.

This I/O request positions the file with the specified DCB identification number to the specified record. A subsequent IN.FREC or OUT.FREC request will read or write that record. The file must be a fixed length record file.

10.6.6. Input Fixed Length Record.

Symbolic name of request code: IN.FREC

Available on: MIKFILE files.

Parameters:

DCB.ID	- DCB identification number for the file from which input is taken.
BUF.ADDR	- Logical address of buffer to receive input. This buffer must have at least the record length of the file and the process must have read/write access to it.

This I/O request reads the next fixed length record.

10.6.7. Output Fixed Length Record.

Symbolic name of request code: OUT.FREC

Available on: MIKFILE files.

Parameters:

DCB.ID	- DCB identification number for the file to which the buffer is output.
BUF.ADDR	- Logical address of buffer to be output.

This I/O request writes the next fixed length record. The number of bytes output is equal to the record length.

10.6.8. Input Variable Length Record.

Symbolic name of request code: IN.VREC

Available on: MIKFILE files.
Terminals.

Parameters:

DCB.ID	- DCB identification number for the unit from which input is taken.
BUF.ADDR	- Logical address of buffer in which the input should be placed.
BUF.LENG	- Length of buffer in which input should be placed.
ECHO.LF	- Applies only to terminals. Specifies if a Line Feed / Carriage Return should be output when the operator presses the return or escape key. Specify 0 for output only Carriage Return, 1 for output Line Feed / Carriage Return. Default: 1.

This I/O request reads the next variable length record from the I/O unit with the specified DCB identification number. If this unit is a terminal, a line of input is read from the terminal.

10.6.9. Output Variable Length Record.

Symbolic name of request code: OUT.VREC

Available on: MIKFILE files.
Terminals.
Printers.

Parameters:

DCB.ID	- DCB identification number for the unit to which the buffer is output.
BUF.ADDR	- Logical address of buffer to be output.
BUF.LENG	- Length of buffer to be output.
OUTP.CNT	- Applies only to terminals and printers. Specifies if a < in the beginning of the buffer denotes the start of a control sequence or should be output directly. Specify 0 for use as control sequence, 1 for output directly. Default: 1.

For MIKFILE files this I/O request writes the specified buffer as the next variable length record onto the file with the specified DCB identification number.

For terminals and printers this I/O request outputs a character string. After this, a Line Feed / Carriage Return will be output, unless specified otherwise in a control sequence. The buffer may contain a control sequence. This is a string of characters delimited by < and >. The < must be the first character in the buffer.

10.6.10. Backwards Input Variable Length Record.

Symbolic name of request code: BIN.VREC

Available on: MIKFILE files.

Parameters:

DCB.ID	- DCB identification number for the file from which input is taken.
BUF.ADDR	- Logical address of buffer in which the input should be placed.

BUF.LENG - Length of buffer in which input should be placed.

This I/O request reads the variable length record preceding the current position of the file with the specified DCB identification number.

10.6.11. Backwards Output Variable Length Record.

Symbolic name of request code: BOU.VREC

Available on: MIKFILE files.

Parameters:

DCB.ID	- DCB identification number for the file to which the buffer is output.
BUF.ADDR	- Logical address of buffer to be output.
BUF.LENG	- Length of buffer to be output.

This I/O request writes the contents of the buffer as a variable length record onto the file area preceding the current position in the file with the specified DCB identification number.

10.6.12. Rename File.

Symbolic name of request code: REN.FILE

Available on: MIKFILE files.

Parameters:

UNIT.NAM	- Name of file to be renamed, including disk specification.

UNIT.NA2 - New name of file, excluding disk specification.

This directive changes the name of the specifed file.

Example:
To change the name of the file ALPHA.T on the disk /DISK01 to BETA.Y, the following parameters should be specified:

```
        DC.W    UNIT.NAM
        DC.B    ´/DISK01/ALPHA.T´,0
        DC.W    UNIT.NA2
        DC.B    ´BETA.Y´,0
```

10.6.13. Delete File.

Symbolic name of request code: DEL.FILE

Available on: MIKFILE files.

Parameters:

UNIT.NAM - Name of file to be renamed, including disk specification.

This directive deletes the specifed file.

10.6.14. Update Character String.

Symbolic name of request code: EDIT.BUF

Available on: MIKFILE files.
Terminals.

Parameters:

DCB.ID	- DCB identification number for the I/O unit from which input is taken.
BUF.ADDR	- Logical address of buffer containing string to be output and in which the input should be placed.
BUF.LENG	- Length of buffer containing string to be output and in which input should be placed.
NOT.MOD	- The number of characters in the beginning of the buffer that may not be modified by the input operation. Default: 0.
CUR.OFFS	- The number of characters, following the non-modifyable characters, that the cursor should be offset when the input operation starts. Irrelevant for MIKFILE files. Default: 0.
OUTP.CNT	- Specifies if a < in the beginning of the buffer denotes the start of a control sequence or should be output directly. Specify 0 for use as control sequence, 1 for output directly. Irrelevant for MIKFILE files. Default: 1.
ECHO.LF	- Specifies if a Line Feed / Carriage Return should be output when the operator presses the return or escape key. Specify 0 for output only Carriage Return, 1 for output Line Feed / Carriage Return. Irrelevant for MIKFILE files. Default: 1.

For terminals this I/O request writes a character string to a terminal, the operator is allowed to edit the string, whereupon the string is read again.

For MIKFILE files this I/O request reads the next variable length record from the I/O unit with the specified DCB identification number. The data is stored in the specified buffer starting at position NOT.MOD.

10.6.15. Define Attention Semaphore

Symbolic name of request code: DEF.ATTN

Available on: Terminals.

Parameters:

UNIT.NAM - Name of I/O unit (terminal) for which the attention semaphore is defined.

ATTN.SEM - Name of attention semaphore. Must belong to the process issuing the directive or be specified as 8 blanks.

Note: With this request the reply semaphore of the IN.OUT system directive is not used. Control returns to the process issuing the request when the operation has been performed.

This request specifies that when the escape key is pressed on the specified terminal (except during input operations) a message should be sent to the specified semaphore. The message will contain 8 bytes specifying the name of the terminal on which the escape key was pressed; for example, ´TERM03 ´.

If the attention semaphore is specified as 8 blanks, no attention message is sent.

10.6.16. Revert Attention Semaphore.

Symbolic name of request code: REV.ATTN

Available on: Terminals.

Parameters:

UNIT.NAM - Name of I/O unit (terminal) for which the attention semaphore is reverted.

Note: With this request the reply semaphore of the IN.OUT system directive is not used. Control returns to the process issuing the request when the operation has been performed.

This request specifies that when the escape key is pressed on the specified terminal (except during input operations) normal operating system action should take place, that is, the > prompt will be output to the terminal.

When a process dies, this request is automatically issued for all terminals for which it has specified attention semaphores.

10.6.17. Enter Direct Input Mode

Symbolic name of request code: ENTER.DI

Available on: Terminals.

Parameters:

UNIT.NAM	- Name of I/O unit (terminal) for which the direct input mode is desired.
BUF.ADDR	- Logical address of buffer to be used by the direct input operation.
BUF.LENG	- Length of buffer used by the direct input operation.
DI.SEM	- Name of direct input semaphore. Must belong to the process issuing the directive.

Note: With this request the reply semaphore of the IN.OUT system directive is not used. Control returns to the process issuing the request when the operation has been performed.

This request specifies that when any key is pressed on the specified terminal (except during normal input operations) a message should be sent to the specified semaphore. The message will contain 8 bytes specifying the name of the terminal on which the key was pressed; for example, ´TERMO3 ´.

Upon receipt of this message the next character may be found in the buffer. This buffer is a cyclic buffer; the first character is stored in the first position, the second character in the second position etc.

10.6.18. Exit Direct Input Mode

Symbolic name of request code: EXIT.DI

Available on: Terminals.

Parameters:

UNIT.NAM - Name of I/O unit (terminal) for which the direct input mode is to be exited.

Note: With this request the reply semaphore of the IN.OUT system directive is not used. Control returns to the process issuing the request when the operation has been performed.

This request causes the specified terminal to revert to normal input operation.

When a process dies, this request is automatically issued for all terminals for which it has specified direct input mode.

10.6.19. Define Function Key Area

Symbolic name of request code: DEF.FKEY

Available on: Terminals.

Parameters:

UNIT.NAM - Name of I/O unit (terminal) for which the function key area is to be defined.

BUF.ADDR - Address of a one byte data area.

This request specifies that whenever a function key is pressed on the specified terminal, the corresponding key value is stored in the specified data area.

10.6.20. Get Catalog Item

Symbolic name of request code: CAT.DISK

Available on: MIKFILE disks.

Parameters:

UNIT.NAM - Name of I/O unit (disk) from which to get catalog item.

FILE.ID - File identification number.

BUF.ADDR - Logical address of buffer in which file name is stored.

BUF.LENG - Length of buffer in which file name is stored.

FILE.IDA - Logical address of long word memory location in which the file identification number of the next file is stored.

This I/O request gets the name of the file with a file identification number following the FILE.ID specified.

To get the name of the ´first´ (in some sense of the word) file on the disk, the FILE.ID should be specified as zero. This will cause the file system to store the name of the first file in the indicated buffer followed by a 0-byte. A file identification number will be stored in the address specified as FILE.IDA. Using this file identification number as the FILE.ID parameter in a subsequent Get Catalog Item request will yield the name and file identification number of the ´next´ (in some sense of the word) file on the disk.

When no more files are found in the catalog, the first byte stored in the buffer will be zero.

10.6.21. Input Sectors.

Symbolic name of request code: IN.SECT

Available on: Disks.

Parameters:

UNIT.NAM -	The name of the disk on which the input operation should take place.
BUF.ADDR -	Logical address of the buffer that is to receive the data input.
BUF.LENG -	Length of input buffer. Must be a multiple of 256 bytes.
STRT.SEC -	Number of first sector to be input from disk.

This I/O request reads a number of sectors from the specified disk. The number of sectors read is the length of the input buffer divided by 256.

10.6.22. Output Sectors

Symbolic name of request code: OUT.SECT

Available on: Disks.

Parameters:

UNIT.NAM -	The name of the disk on which the output operation should take place.
BUF.ADDR -	Logical address of the buffer containing the data to be output.
BUF.LENG -	Length of output buffer. Must be a multiple of 256 bytes.
STRT.SEC -	Number of first sector to be input from disk.

This I/O request writes a number of sectors to the specified disk. The number of sectors written is the length of the input buffer divided by 256.

10.6.23. Set Program Privilege Status.

Symbolic name of request code: SET.PRIV

Available on: MIKFILE files.

Parameters:

UNIT.NAM - Name of file for which the privilege status should be changed.

PRIV.VAL - 0 means remove privilege status.
1 means set privilege status.

This request changes the privilege status of a load module. The program contained in the specified file will be executed as a privileged process when loaded if PRIV.VAL is specified as 1.

This request may only be issued by privileged processes.

10.6.24. Get Load Module Information.

Symbolic name of request code: GET.LOAD

Available on: MIKFILE files.

Parameters:

UNIT.NAM - Name of file containing load module. If file type is not specified, ´.1´ will be added to the file name.

BUF.ADDR - Logical address of a word-aligned ?-byte data area in which the load module information will be stored.

>> The layout of the information has not yet been determined <<

This request fetches information about an executable program.

10.6.25. Load Load Module.

Symbolic name of request code: LOAD.MOD

Available on: MIKFILE files.

Parameters:

UNIT.NAM - Name of file containing load module. If file type is not specified, ´.1´ will be added to the file name.

SEG.ADDR - 16 physical (!) addresses of the memory locations into which the program segments should be loaded.

This request loads an executable program into memory. The request may only be issued by privileged processes, and should normally only be executed by the supervisor.

10.6.26. Close All Files.

Symbolic name of request code: CLO.ALL

Available on: MIKFILE files.

Parameters:

PROC.NAM - Name of process.

This request causes all files belonging to the specified process to be closed. Only privileged processed may close files opened by processes belonging to other users, and this should normally only be done by the supervisor.

11. Logical I/O units.

All I/O units in the system may be referred to as specified in section 10. However, it is often desired to use a so-called ´logical unit´ for part or all of the I/O unit specification. A logical unit is simply a string variable of up to 8 characters used instead of part or all of the I/O unit specification. The operating system maintains for each process a list of logical units.

Logical units are specified enclosed in < and >.

Examples:

If the logical unit <ALPHA> is assigned the value ´!COM1/´, the logical unit <BETA> is assigned the value ´!COM1/DISK01/´, and the logical unit <GAMMA> is assigned the value ´!COM1/DISK01/FILE1.K´, the I/O unit specification ´!COM1/DISK01/FILE1.K´ may alternatively be given as ´<ALPHA>DISK01/FILE1.K´, ´<BETA>FILE1.K´, or simply ´<GAMMA>´.

If the logical unit <DELTA> is assigned the value ´/´, the logical unit <EPSILON> is assigned the value ´/DISK01/´, and the logical unit <ZETA> is assigned the value ´/DISK01/FILE1.K´, the I/O unit ´/DISK01/FILE1.K´ may be reffered to as ´<DELTA>DISK01/FILE1.K´, ´<EPSILON>FILE1.K´, or simply ´<ZETA>´.

A logical unit may only replace the first part of an I/O unit specification.

The logical unit assignments used with a process are determined in the following manner:

First, a set of standard assignments are given:

<SYSDISK>	=	/DISK01/	(disk number 1)
<LIST>	=	/PRINT01	(printer number 1)
<MYTERM>	=	/TERMnn	(the terminal where the program was started)

Second, the user may set up assignments that are in effect for all programs started on the terminal until the user logs off. These assignments will override the standard assignments.

Third, the user may, when starting a process, set up assignments that pertain to that particular process only. These assignments will override any assignments given above. A description of how to specify these assignments when one process starts another one is found in section 6.6.

Fourth, a process may set up assignments for itself. These assignments will override any assignments given above.

When one process starts another one, the started process will inherit all assignments pertaining to the starting process, unless these assignments are overridden in the execute or spawn directive.

11.1. System Directives.

The following directives exist:

11.1.1. Assign Globally.

Symbolic name of directive number: ASS.GLOB

Data block:

DC.B	logical unit (8 bytes)
DC.B	character string which is to be assigned to the logical unit,0

This directive specifies a logical unit assignment which will be in effect until the logical unit is assigned another value or the user logs off.

This assignment will pertain to all processes started on the terminal after this directive has been issued.

A logical unit may be deleted by specifying the second parameter in the data block as simply 0.

11.1.2. Assign Locally.

Symbolic name of directive number: ASS.LOC

Data block:

DC.B	logical unit (8 bytes)
DC.B	character string which is to be assigned to the logical unit,0

This directive specifies a logical unit assignment which will be in effect until the logical unit is assigned another value or the process issuing the directive terminates.

A logical unit may be deleted by specifying the second parameter in the data block as simply 0.

This assignment will be local to the process issuing the directive and any process it may execute or spawn.

11.1.3. Get I/O Unit.

Symbolic name of directive number: GET.IOU

Data block:

DC.B	I/O unit specification,0
DC.W	maximum allowable length in bytes of the I/O unit ex-

	pansion
DC.L	logical address of a word-aligned data area of at least the length specified in the second parameter given in this data block. The process issuing the directive must have read/write access to the data area.

This directive expands the I/O unit specification given as the first parameter in the data block and stores this expansion followed by a 0-byte in the data area whose address is given as the last parameter in the data block.

The I/O unit specification will generally contain a logical unit specification.

If the I/O unit expansion takes up more bytes than specified as the maximum allowable length, it will be truncated to that length.

12. Exception Handling.

A user process may supply an ´exception handler´. This is formally a subroutine (return is made via the RTR instruction). This subroutine is called whenever a particular exception arises.

The user may supply exception handlers for the following exceptions:

Exception number	Exception name
2	Bus error
3	Address error
4	Illegal instruction
5	Zero divide
6	CHK instruction
7	TRAPV instruction
8	Privilege violation
9	Trace
10	Line 1010 emulator
11	Line 1111 emulator
32-46	TRAP 0 - TRAP 14

The supervisor has a standard set of exception handlers causing a fatal error in the program causing the exception.

The following system directive exists:

12.1. Define Exception Handler.

Symbolic name of directive number: DEF.EXC

Data block:

DC.B	exception number
DC.L	logical address of exception handler

This directive declares that a user-written excpetion handler be used executed with the trace bit set in the contents given by the condition codes.

After the execution of the specified instruction the new value of the program counter and the condition codes are saved on the stack and execution continues in the trace exception handler. Return from the handler is made by issuing the Enter Trace State directive again or by a RTR instruction, in which case tracing will be turned off.

If the instruction causing the trace exception also caused another exception, such as Zero Divide, for which a user-written exception handler exists, the trap exception handler will be entered first, and the return address will be that of the Zero Divide exception handler.

when the specified exception occurs. This directive should not be used with the trace exception.

When the exception occurs, the supervisor ensures that the execution of the user program proceeds from the specified logical address. The contents of the stack is the same as when the supervisor was called, but execution takes place in the user state. The user-written exception handler is turned off when the exception occurs, and the user must issue a new directive to turn it on again. Thus if a Zero Divide exception occurs during the handling of a Zero Divide exception the normal supervisor handling of the exception will take place unless the user has already re-defined the exception handler.

13. Miscellaneous System Services.

13.1. Get System Time.

Symbolic name of directive number: GET.TIME

Data block:

DC.L	logical address of a 20 byte data area to which the process issuing the directive has read/write access. The layout of this data area is:

DS.B	10	here the date will be stored in ASCII in the format ´dd.mm.yyyy´.
DS.B	8	here the time will be stored in ASCII in the format ´hh.mm.ss´.
DS.W	1	here the day-of-week will be stored as an integer ranging from 1-7 with 1 signifying Monday, 2 signifying Tuesday, ..., 7 signifying Sunday.

This directive fetches the system time and stores it in the indicated data area.

13.2. Set System Time.

Symbolic name of directive number: SET.TIME

Data block:

DC.B	10 bytes containing the date in ASCII in the format ´dd.mm.yyyy´.
DS.B	8 bytes containing the time in ASCII in the format ´hh.mm.ss´.

This directive sets the system date and time to the specified values.

If either the date or the time is specified as blank characters, the current value of this item is not updated.

This directive may only be issued by privileged processes.

13.3. Inter-process Move.

Symbolic name of directive number: IP.MOVE

Data block:

DC.B	name of source process (8 bytes)
DC.W	user number of source process
DC.L	logical address in source process of block to be moved
DC.B	name of destination process (8 bytes)
DC.W	user number of destination process
DC.L	logical address in destination process of block to receive data
DC.L	number of bytes to move

This directive moves the specified number of bytes from the indicated location in the source process to the indicated location in the destination process. Low byte is moved first.

Either or both process names may be specified as 8 blanks, in which case the process issuing the directive is assumed.

Either or both of the user numbers may be specified as $FFFF, in which case the user number of the process issuing the directive is assumed.

Unprivileged processes may only access memory locations to which they have read-only or read/write access, as appropriate.

13.4. Get Hardware Configuraton.

Symbolic name of directive number: GET.HW

Data block:

DC.L	Logical address of a word-aligned 2-byte data area to which the process issuing the directive has read/write access.

This directive stores information about the hardware configuration of the computer in the specified memory location.

Appendix A. Error Codes.

This appendix list the error codes returned by the supervisor or MIKFILE file system.

Symbolic name	Hex value	Description
E.IDIR	$FFFF	Illegal directive number.
E.NNAME		Name of process could not be generated.
E.IEIT		Illegal EIT.
E.ILUN		Illegal logical unit name.
E.IPRNAM		Illegal process name.
E.INOLU		Illegal number of logical unit assignments.
E.MEMVIO		Memory access violation.
E.ISMNAM		Illegal semaphore name.
E.NATTSM		Process not attached to semaphore.
E.AATTSM		Process already attached to semaphore.
E.IPGNAM		Illegal name of installed program.
E.ICPU		Illegal CPU number.
E.AINPRG		Program already installed.
E.NINPRG		Program not installed.
E.PRIVIO		Privilege violation.
E.NXSTPR		Process does not exist.
E.AXSTPR		Process already exists.
E.NWORDA		Memory address not word-aligned.
E.AXSTSM		Semaphore already exists.
E.IMESLN		Illegal message length.
E.MSLONG		Message too long.
E.IUSENO		Illegal user number.
E.ARESRV		Resource already reserved.
E.NRESRV		Resource not reserved.
E.IRSKND		Illegal reservation kind.
E.IPANAM		Illegal partition name.
E.NATTPA		Process not attached to partition.
E.AATTPA		Process already attached to partition.
E.AXSTPA		Partition already exists.
E.PALONG		Partition too long.
E.SEGUSE		Segment already in use.

E.ISEGM	Illegal segment.
E.IACCOD	Illegal access code.
E.NACCPA	Partiton may not be accessed in the desired manner.
E.IDCBID	Illegal DCB id.
E.ICONAM	Illegal computer name.
E.CNMNET	Computer not in MikNet.
E.DVNCOM	I/O device not in computer.
E.IOAVIO	I/O unit access right violation.
E.IIOREQ	Illegal I/O request code.
E.IPARAM	Illegal parameter type.
E.IACLEV	Illegal access level.
E.NFILES	File system not imposed on disk.
E.AFILES	File system already imposed on disk.
E.IEXCNO	Illegal exception number.
E.IDATIM	Illegal date or time.
E.IBYTES	Illegal number of bytes in move.
E.RESUME	Execution was resumed by ´Resume´ directive.
E.NXSTFL	File does not exist.
E.AXSTFL	File already exists.
E.DSKFUL	Disk full.
E.IRECLN	Illegal record length.
E.FILUSE	File in use.
E.FILNOP	File not open.
E.FILNCL	File not closed.
E.EXT60	File extended 60 times.
E.IFLNAM	Illegal file name.
E.IDSKID	Illegal disk id.
E.IRECNO	Positioning to non-existent record.
E.RECOUT	Access to record that lies partly or completely outside the file boundaries.
E.FRONLY	File open for read only.
E.CATFUL	Disk catalog full.
E.IDCBLN	Illegal DCB length.
E.ISECNO	Illegal number of sectors in file.
E.IFILTP	Illegal file type.
E.RECLMM	Record length mismatch.
E.DNOTRD	Disk not ready.
E.HARDER	Hard error on disk.

E.DRONLY	Disk is write protected.
E.ITSNO	Illegal track/sector number or illegal buffer length.
E.TRLONG	Transfer extends past last sector of disk.
E.IPRIO	Illegal priority.
E.NOMEM	No memory available.
E.IENTRY	Illegal program entry point.
E.ILOADM	Illegal load module file structure.