

GenRef
v1.00

MDOS Reference guide.

Memory Library

(C) Copyright 1989
J. Paul Charlton
ALL RIGHTS RESERVED

MEMORY CONTENTS

	Page
Memory management overview.....	1
Calling memory functions.....	2
Available memory.....	3
Allocate memory.....	3
Release memory.....	9
Map memory.....	12
Get memory list.....	14
Declare shared memory.....	16
Release shared memory.....	18
Use shared memory.....	19
Get size of shared group.....	20
Free task.....	21
Get memory page.....	21
Free memory page.....	22
Free memory node.....	23
Link memory node.....	23
Get memory list (system).....	24

MEMORY MANAGEMENT OVERVIEW

The memory management routines in MDOS are provided to aid a programmer in writing applications which are larger than the 64 Kbytes directly addressable by the CPU's 16 address lines. They also serve the purpose of providing each task with it's own private address space, separate from memory accessible to other tasks.

Each task under MDOS can have 2 Mbytes of virtual memory, using 21 address bits. The 21 address bits consist of two fields. The first field includes the eight most significant address bits, and is referred to as the virtual page number. The second field consists of the thirteen least significant address bits, and is referred to as the page offset.

The physical memory in the Geneve computer has 21 address lines, for a maximum of 2 MBytes of physical memory. Like virtual memory, each physical address can be thought of as a 21 bit address of two fields, with the first eight bit field called the physical page number, and the final thirteen bit field referred to as the page offset.

NOTE: It can be easy to confuse physical pages with virtual pages, so be careful when reading the opcode descriptions below.

The 16 address lines provided by the 9995 processor can be thought of two fields. The first field is the most significant three bits of the address and is called the "window number". The least significant thirteen bits are the page offset. These 16 address bits can be referred to as the linear address space of the CPU.

The virtual address space and the linear address space are of the most interest to people writing tasks which run under MDOS. The memory management routines provide transparent methods of assigning physical memory pages into your task's virtual address space and transparent methods of viewing any 8k block of a task's virtual address space within one of the seven usable 8k memory windows in the linear address space.

MDOS maintains two arrays to manage the physical memory pages belonging to your task. The first array, which only contains 8 physical page numbers, is part of the Geneve hardware and is called the "mapper". The mapper is used to assign a physical memory page to each of the eight 8k windows addressable in the 9995's linear address space. The second array, which is actually stored as a singly linked list within MDOS, is stored in RAM under control of the MDOS memory management routines and is referred to as "the virtual page list". Each node in the virtual page list consists of a physical page number and various attributes for that page. Various attributes used in the virtual page list allow for pages to be unassigned (correspond to no useful physical page), for pages to be shared, for pages to be disk-resident (swapped out), and for pages to be private (accessible to only your task.)

CALLING MEMORY FUNCTIONS

The MDOS memory management functions must be called from within a machine code program running as a task under MDOS. You pass arguments to the memory management functions using only a few registers of your program's workspace.

The MDOS memory management functions are invoked from a machine code program when software trap number zero (XOP 0) is called with a library number of 7. The calling program's R0 must contain the opcode of the routine within the memory management library which is to be performed. The following code fragment will allocate memory to your task.

```

                LI      R0,1
                LI      R1,7      56k bytes
                LI      R2,1      starting @>2000
                SETO    R3        try fast pages
                XOP     @SEVEN,0
                MOV     R0,R0
                JNE     MEMERR
*   ...
SEVEN  DATA      7
*   ...

```

AVAILABLE MEMORY

Function You would use this operation in a program when you want to determine how much memory is available for use. It returns the total number of 8k pages installed, the number of zero wait state 8k pages available, and the total number of 8k pages available (both fast and slow pages).

Parameters R0 = 0 (opcode)

RESULTS R0 = 0 (no error)
 R1 = total number of free pages
 R2 = number of free zero wait state pages
 R3 = total number of installed pages

ALLOCATE MEMORY

Function This routine allows you to assign physical pages of memory from system list of free pages to virtual pages belonging to your task. You must use this function if you wish to use more memory than your program occupied on disk as a program image file. You must also use this function if you wish to use more than 64k of memory in a program you have written. This routine will not reassign pages which have already been allocated, even if the block of pages you specify overlaps pages which have already been assigned to your task.

On successful return, all pages in the range R2..(R2+R1-1) are available for use by your task.

Parameters R0 = 1 (opcode)
 R1 = page count
 R2 = starting page
 R3 = speed flag 0 (use first available memory page)
 <>0 (use zero wait state pages, if available)

Results R0 = error code
 R1 = new count
 R2 = fast count

Parameter Description

Page count This is the number of consecutive memory pages you wish to have for your program, it is not necessarily the number of pages which will be returned to your program. As an example, to allocate 20k bytes of memory for your program, you must actually ask for three 8k memory pages. The number of pages you need to ask for can be calculated from the number of bytes you need as follows:

$$\text{pages} = (\text{bytes} + \text{>1fff}) / \text{>2000}$$

Starting page This is the virtual page number within your task's memory at which you want to allocate more memory pages. If you were to think of your task's memory as having addresses ranging from >000000 to >1fffff (0 to 2 MB), this number is the address divided by 8192 (remainder is discarded.)

Speed flag If this flag is non-zero, MDOS will attempt to assign zero wait state memory pages to your task. If there are not enough zero wait state pages available to satisfy your request, MDOS will assign slow pages to your task in order to satisfy the request.

If this flag is zero, and your computer has 512k of (one wait state) RAM on the motherboard, MDOS will first attempt to assign slow pages to your task. If there are not enough slow pages, MDOS will continue by allocating fast pages to your task.

If this flag is zero, and your computer has 1024k of (zero wait state) RAM on the motherboard, MDOS will first attempt to assign fast pages to your task. If there are not enough fast pages, MDOS will continue by allocating slow pages to your task.

The "fast count" returned to you reflects the number of fast pages allocated as a result of the operation, and the "fast count" subtracted from the "new count" returned to you reflects the number of slow pages allocated as a result of the operation.

If the "fast count" returned to you is non-zero, and different from the number of pages you requested, there is no convenient method of determining which pages are fast, and which are slow. The easiest deterministic method of telling which pages are fast and which are slow is to ask

MDOS for one page at a time, and look at the "fast count" resulting from each single-page allocation.

Error code

0 = No error. This indicates that the pages you specified can now be used by your task.

1 = Insufficient memory. When you get this error, there were not enough pages free in the system to accommodate your request for more memory. No additional pages have been assigned to your task, even if there were some free memory pages in the system.

(NOTE: Calling the "Available memory" operation to determine the amount of memory available, followed by the "Allocate memory" operation with fewer pages than reported to you from "Available memory" can still fail, since another task may have allocated pages in between your two calls. Do not rely on being able to call the two routines in succession without checking the error code returned from the "Allocate memory" operation.)

7 = Attempt to overwrite shared page. You will get this error if any page in the range $R2..(R2+R1-1)$ is already allocated to your task with a "shared" attribute. No additional pages have been assigned to your task if you receive this error, even if there were enough free memory pages in the system to accommodate your request.

8 = Out of table space. You will receive this error if too many tasks have large gaps of unassigned pages in their memory maps. The current versions of MDOS allow 480 virtual pages between all tasks which are currently executing. Note that there are only 256 possible physical pages, and that there are only 128 physical pages even if you have the 512k expansion RAM, so tasks would have to be pretty wasteful (have more gaps than actual pages) in order to use up all 480 virtual pages allowed by MDOS. If you get this error, your program should just give up and tell the user to try later.

New count

This is the number of pages which were newly assigned to your task, and is only valid if you did not receive an error from the "allocate memory" call. This number can be less than the number of pages you requested if some of the pages in the range $R2..(R2+R1-1)$ were already assigned to your task.

Fast count

This is the number of fast pages which were newly assigned to your task, and is only valid if you did not receive an error from the "allocate memory" call. You would use this to check if MDOS actually assigned any fast pages to your task.

Example 1.1**Filling a hole**

R0=1 Opcode
 R1=2 Number of pages to get
 R2=4 Virtual page number
 R3=0 Speed flag

Virtual Page	Physical	Physical	
	Page	Page	
	Before	After	
0	>3F	>3F	
1	>3E	>3E	
2	>3D	>3D	
3	(>FF)	(>FF)	hole
4	(>FF)	>33	new page
5	(>FF)	>32	new page
6	(>FF)	(>FF)	hole
7	>3C	>3C	
8	>3B	>3B	
9	>3A	>3A	
10	>39	>39	
11	>38	>38	
12	>37	>37	
13	>36	>36	
14	>35	>35	
15	>34	>34	

Note:

The pages (>FF) represent holes in the tasks virtual memory map. The physical page >FF is actually part of the boot rom on your computer, and cannot be overwritten by your task.

Example 1.2

R0=1	Opcode		
R1=1	Number of pages to get		
R2=17	Virtual page number		
R3=0	Speed flag		
Virtual	Physical	Physical	
Page	Page	Page	
	Before	After	
0	>3F	>3F	
1	>3E	>3E	
2	>3D	>3D	
3	(>FF)	(>FF)	hole
4	(>FF)	(>FF)	hole
5	(>FF)	(>FF)	hole
6	(>FF)	(>FF)	hole
7	>3C	>3C	
8	>3B	>3B	
9	>3A	>3A	
10	>39	>39	
11	>38	>38	
12	>37	>37	
13	>36	>36	
14	>35	>35	
15	>34	>34	
16	(null)	(>FF)	new hole
17	(null)	>33	new page

Notice that this routine only **fills** holes, it does not assign a new physical page to a virtual page which is already assigned to your task.

Example 1.3**Overlaying Pages**

R0=1 Opcode
 R1=5 Number of pages to get
 R2=1 Virtual page number
 R3=0 Speed flag

Virtual Page	Physical	Physical	
	Page	Page	
	Before	After	
0	>3F	>3F	
1	>3E	>3E	no change
2	>3D	>3D	no change
3	(>FF)	>33	new page
4	(>FF)	>32	new page
5	(>FF)	>31	new page
6	(>FF)	(>FF)	hole
7	>3C	>3C	
8	>3B	>3B	
9	>3A	>3A	
10	>39	>39	
11	>38	>38	
12	>37	>37	
13	>36	>36	
14	>35	>35	
15	>34	>34	

Notice that even though you asked for 5 pages, only 3 were actually assigned, since two of the specified pages had already been assigned.

RELEASE MEMORY

Function You will use this routine to return unused memory to MDOS, this is useful if your program uses lots of temporary data. This is also one of the functions used by MDOS to free memory when your task is terminated. Any page which is released by your task which is also currently mapped into one of your task's seven memory windows will be removed from the memory window used by your task, and its entry in the mapper will be replaced by page >FF.

You may not release virtual page zero of your task using this function (although page zero may be accessed by your task, it doesn't really belong to your task.)

This opcode cannot be used to free shared pages belonging to a task. Shared memory pages must be freed with opcode #6.

Parameters R0 = 2 (opcode)
 R1 = page count
 R2 = starting page

Results R0 = error code

Parameter description

Page count This the number of memory pages you wish to free from your program. It is not necessarily the same as the number of physical memory pages which will actually be freed from your task. Shared pages, and unallocated pages in the range R2:(R2+R1-1) will not be released from your task. No pages will be released if this count is zero.

Starting page This is the page number of the first virtual memory page you wish to have released from your task. This procedure will attempt to release all of your task's virtual memory pages in the range R2:(R2+R1-1) into the free page pool.

Error code 0 = No Error. This indicates that the non-shared pages in the range specified have been released from your task back to the free pages in the system.

2 = Attempt to free page zero. This indicates that you tried to free virtual page zero of your task. No pages were actually released from your task.

8 = Out of table space. MDOS was unable to free a page because there weren't enough virtual pages nodes available to create a new page in the free pool. When you receive this error, it is possible that some, but not all, of the pages in the range R2:(R2+R1-1) have been moved to the free pool. You will receive this error if too many tasks have large gaps of unassigned pages in their memory maps. The current versions of MDOS allow 480 virtual pages between all tasks which are currently executing. Note that there are only 256 possible physical pages, and that there are only 128 physical pages even if you have the 512k expansion RAM, so tasks would have to be pretty wasteful (have more gaps than actual pages) in order to use up all 480 virtual pages allowed by MDOS. If you get this error, your program should just give up and tell the user to try later.

Example 2.1**Making a Hole**

```
R0=2      Opcode
R1=9      Number of pages to release
R2=2      First virtual page to release
```

Virtual Page	Physical Page		
	Before	After	
0	>3F	>3F	
1	>3E	>3E	
2	>3D	(>FF)	new hole
3	(>FF)	(>FF)	hole
4	(>FF)	(>FF)	hole
5	(>FF)	(>FF)	hole
6	(>FF)	(>FF)	hole
7	>3C	(>FF)	new hole
8	>3B	(>FF)	new hole
9	>3A	(>FF)	new hole
10	>39	(>FF)	new hole
11	>38	>38	
12	>37	>37	
13	>36	>36	
14	>35	>35	
15	>34	>34	

Note that only five pages were actually released from your task to MDOS, since some of the pages in the specified range were already unassigned.

Example 2.2**Making list shorter**

R0=2 Opcode
 R1=8 Number of pages to release
 R2=10 First virtual page to release

Virtual Page	Physical	Physical	
	Page	Page	
	Before	After	
0	>3F	>3F	
1	>3E	>3E	
2	>3D	>3D	
3	(>FF)	(>FF)	
4	(>FF)	(>FF)	
5	(>FF)	(>FF)	
6	(>FF)	(>FF)	
7	>3C	>3C	
8	>3B	>3B	
9	>3A	>3A	
10	>39	null	freed
11	>38	null	freed
12	>37	null	freed
13	>36	null	freed
14	>35	null	freed
15	>34	null	freed

The list was truncated, since all of the pages at the tail of the list were unassigned. Also note that we really told it to release pages 10 to 18, but we only had pages up to 15 to begin with. No error is reported when you attempt to release unassigned pages.

MAP MEMORY

Function This routine can be used to place a physical page into the mapper chip for the specified virtual page belonging to your task. You can think of the mapper as providing seven usable 8k "memory windows" into your task's virtual memory space. You tell this routine which of the seven windows to use, and which part of virtual memory to look at. This routine can not be used to overwrite page zero of your task.

You should use this routine for mapping memory if you want your program to be able to use transparent demand paging in future versions of MDOS which support page swapping to hard disk.

(NOTE: If you are using window number seven, the one which is at >E000 in your direct address space; the data from offset >1000 to >1140 in the page will be corrupted by writes to addresses in the range >F000 to >F140. Do not use window number seven unless it is ok for the data in the specified range to be corrupted.)

On successful return, the specified virtual page belonging to your task has been mapped into the window you specified and is available for use.

Parameters

R0	= 3 (opcode)
R1	= page number
R2	= window number

Results

R0	= error code
mapper	= new page

Parameter description

Window number This parameter, in the range 1:7, is used to tell MDOS which of the seven 8k byte windows in the processor's 16-bit address space to use for the specified virtual page belonging to your task.

Processor address = Window_number * >2000

Page number This is the virtual page number within your task that you wish to have mapped into the specified window in the processor's 16-bit address space. Virtual memory in the address range (page * >2000):(page * >2000 +>1fff) will be accessible to your program with the 16-bit addresses (window * >2000):(window * >2000 + >1fff).

Error code 0 = No Error. This indicates that the specified virtual memory page of your task has been mapped into the specified memory window.

 2 = Header page mapping violation. You attempted to map a virtual page into window zero, which is reserved for your task's header by MDOS. Alternatively, you attempted to map virtual page zero, your task's header, into some other memory window. Your task's memory map has not been changed if you get this error.

 3 = Unassigned virtual page, or Invalid memory window. The virtual page you specified has never been allocated by your task, and contains no valid data (it is a "hole" in your address space.) Alternatively, you specified a window number larger than seven. Your task's memory map has not be changed if you get this error.

Mapper In MDOS mode, the mapper is 8 bytes long, and located at >F110 in the processor's 16-bit address space. Each byte in the mapper contains a physical memory page number, in the range >00:>FF. Assignments are as follows:

 Mapper register = >F110 + Window_number

 Note that each mapper register corresponds to a specific 8k block in the processor's 16-bit address space. After successful completion of the "Map Memory" function, the mapper register corresponding to the window you specified will contain the physical page number of the virtual page you specified.

 Symbolically:

 mapper[window number] = task_pages[virtual page number]

GET MEMORY LIST

Function

This operation returns an array of physical page numbers (each is one byte) corresponding to the virtual pages belonging to your task. You are allowed to specify the address of the first byte in the array, and the maximum number of elements in the array.

This array of physical page numbers is useful if you need to speed up memory paging in your task by use of your own paging code. If your program performs its own paging, you will need to call this opcode after every call to a memory management function which adds or removes pages from your virtual address space. (Opcodes 1,2,6,7)

In future versions of MDOS which support transparent demand paging, this operation will also "lock" **all** of your task's virtual pages into RAM, making them ineligible for paging. To "unlock" your virtual pages, another opcode will be provided for your use. (Normally, only pages which are currently mapped into one of the seven processor windows for your task would be "locked" into RAM.)

On successful return, the array will contain the physical page numbers for the virtual pages in your task.

Parameters

R0 = 4 (opcode)
R1 = array start
R2 = array size

Results

R0 = error code
R1 = array used

Parameter description

Array start

You specify the address of the first byte in your array using this parameter. The physical page number of your task's header page, virtual page number zero, is placed in the first byte of the array. This is a 16-bit processor address for a location which is currently mapped into your task's memory windows.

Array size

This is the maximum number of physical page numbers which can be returned to your task. The indexes of the array elements can range from zero to (array size)-1.

Error code 0 = No Error. This indicates that the array contains all of the physical page associations for the virtual pages in your task. The number of actual array elements used can be less than the maximum size you specified for the array.

 8 = Array not large enough. Your array was not large enough to hold all of the physical page numbers in use by your task. When you get this error, the contents of the array are valid up to the maximum element which you allowed.

Array used This indicates the number of valid pages returned in the array. When you perform your own paging, you should make sure that you never index into the array after the last valid page (You will end up mapping a page which doesn't belong to your task.)

Sample Code

Assuming that you've already called this opcode, the follow code fragment will map in a data item pointed to by a 32 bit address.

```

assume: r1,r2 = 32 bit pointer, @paglst are bytes from opcode #4

movb r2,r1          ok, since only low 5 bits of r1 are used
andi r1,>e01f       keep 8 bits, zap the others
src r1,13           rotate to make an index into the page list
andi r2,>1fff       mask off the high three bits, they're now
*                  in R1 ...
*
movb @paglst(r1),@mapper+4      put it at >8000
movb @paglst+1(r1),@mapper+5    put next page at >a000
*
* it is not necessary to place two pages into the mapper if you
* know for certain that the record accessed by the pointer does
* not cross page boundaries, the above code is just a method of
* playing it safe
*
mov @>8000+field_offset(r2),r3
*
* this of course assumes that there is some record addressed by
* the initial pointer, and that the record contains fields of some
* data structure. Fields are easy to set up with a DORG statement
* for each record type in use by an application
*

```

DECLARE SHARED MEMORY

Function

This routine is used to declare a range of pages currently belonging to your task as "shared" memory pages, which means that they can be used by other tasks. An example of two tasks sharing memory would be an editor which "shared" all of its text buffer with an assembler, so that the assembler could assemble from RAM rather than from disk.

Each group of pages declared as shared has a type, which you assign. When another application wants to share those memory pages with your task, it will ask MDOS to use a certain type of shared pages. An editor buffer could be declared as one specific type, while object code would be declared as a separate type, so that programs would not use the wrong sort of data as input (You wouldn't want a Fortran compiler to use a binary program as its input!)

It is recommended that "types" be assigned by the distributor of MDOS, so that incompatible applications do not try to use the same "type" if you decide to use a "type" please correspond with the distributor of MDOS to coordinate your development efforts with others.

A "shared" type may only be declared once, and always resides in a group of consecutive virtual pages. If all applications using a "shared" group of pages release those pages, the "type" may be redeclared. (MDOS keeps a count of the number of applications using a shared group, and if the count ever becomes zero, the type is made free for re-use)

Note: It is not possible to declare page 0 to be part of a shared group. Page 0 is **always** private, since it contains the information which MDOS uses to distinguish between tasks.

Parameters

R0 = 5 (opcode)
R1 = page count
R2 = starting page
R3 = shared type

Results

R0 = error code

Parameter Description

Page count	This is the number of consecutive virtual pages belonging to your task which will be declared as a shared page group for use by other tasks.
Starting page	This is the virtual page number within your task of the first virtual page which will become part of the shared page group. Pages in the range (start_page):(start_page+page_count-1) will belong to the group.
Shared type	This must be in the range >01:>FE, and should be a code unique to the format of data which you are sharing with other tasks. It is recommended that you use a common set of source code routines for data access for all of your tasks which use the data.
Error code	<p>0 = No Error. This indicates that virtual pages you specified can now be shared by other tasks running under MDOS.</p> <p>3 = Bad page. At least one of the pages in the range (start_page):(start_page+page_count-1) has never been allocated by your task. The shared group has not been defined if you get this error.</p> <p>5 = Invalid type code. Your "shared type" parameter was not in the range >01:>FE, or your "shared type" code has already been declared by another task. The shared group does not contain the pages you specified if you get this error.</p> <p>7 = Invalid page declaration. At least one of the pages in the range (start_page) : (start_page + page_count-1) is unallocated, already declared as shared, or is virtual page number zero. The shared group has not been defined if you get this error.</p> <p>8 = Out of table space. MDOS was unable to create the shared type because weren't enough virtual pages nodes available to create a shared page group descriptor list. You will receive this error if too many tasks have large gaps of unassigned pages in their memory maps. The current versions of MDOS allow 480 virtual pages between all tasks which are currently executing. Note that there are only 256 possible physical pages, and that there are only 128 physical pages even if you have the 512k expansion RAM, so tasks would have to be pretty wasteful (have more gaps than actual pages) in order to use up all 480 virtual pages allowed by MDOS. If you get this error, your program should just give up and tell the user to try later.</p>

RELEASE SHARED MEMORY

Function This operation removes all shared memory pages of the specified type from your task's virtual memory list. If your task was the only task using the shared page group, the group will become undefined, and must be redeclared before use. Any page which is released by your task which is also currently mapped into one of your task's seven memory windows will be removed from the memory window used by your task, and its entry in the mapper will be replaced by page >FF.

Parameters R0 = 6 (opcode)
R1 = shared type

Results R0 = error code

Parameter description

Shared type This is a shared group type number, in the range >01:>FE, and must have been previously defined by another task.

Error code 0 = No Error. This indicates that all of the pages from the shared group you specified have been released from your task.

6 = Invalid type. The type you specified was not in the range >01:>FE, or hasn't yet been declared by another task.

8 = Out of table space. MDOS was unable to free a page because there weren't enough virtual pages nodes available to create a new page in the free pool. When you receive this error, it is possible that some, but not all, of the pages belonging to the shared group have been moved to the free pool. If you get this error, your program should just give up and tell the user to try later.

USE SHARED MEMORY

Function This operation will include the pages from the shared type specified in your task's list of virtual pages. The shared type must be been previously declared by another task. When you call this function, all pages in the range (start_page):(start_page+shared_size-1) must not be allocated by your task, since you are not permitted to overlay shared and existing pages in your virtual page list.

Parameters

R0	=	7 (opcode)
R1	=	shared type
R2	=	start page

Results R0 = error code

Parameter Description

Shared type This is a type code for a shared page group which must have been declared by another task. If your task has enough contiguous available virtual pages beginning with the start page you specified, all of the pages from the shared page group will be mapped in at the specified virtual page address.

Start page This is the virtual page number within your task of the first virtual page which will be used by the shared page group. After calling this operation, you must explicitly map in the virtual pages which have just been assigned, since they will not be automatically placed into your task's mapper registers.

Error code 0 = No Error. The shared page group of the type you requested was already defined and was successfully mapped into your task's virtual page list.

2 = Attempt to overlay page zero. You specified virtual page zero as the start page for the shared memory group. No pages from the memory group have been allocated to your task if you get this error.

6 = Invalid shared type. The type you specified was not in the range >01:>FE or has not yet been defined for use by another task. No pages from the memory group have been allocated to your task if you get this error.

7 = Attempt to overlay shared and private memory. Your task did not have enough contiguous free virtual pages starting with the virtual page specified to map in the pages from the shared group. No pages from the memory group have been allocated to your task if you get this error.

8 = Out of table space. There were not enough free nodes available to extend your task's virtual page list. MDOS is out of table space. At this point, your task should give up and tell the user to try later. No pages from the memory group have been allocated to your task if you get this error.

GET SIZE OF SHARED GROUP

Function This operation reports the number of pages which belong to a shared page group. It should be used by your task before you have the shared page group assigned into your virtual page list, so that you know in advance if your task has enough unused contiguous pages to overlay the shared page group.

Parameters R0 = 8 (opcode)
R1 = shared type

Results R0 = error code
R1 = shared size

Parameter description

Shared type This is a type code for a shared page group which must have been declared by another task. On successful return, the number of pages in this page group is returned to your task.

Error code 0 = No Error. The shared page group of the type you requested was already defined and its size was returned to you.

6 = Invalid shared type. The type you specified was not in the range >01:>FE or has not yet been defined for use by another task.

Shared size On successful return, this will contain the size, in 8k pages, of the specified shared page group.

FREE TASK

Function This routine can not be directly used by tasks under MDOS, it is reserved for use by system library functions.

This is used to free all memory pages, except the task's header, from the task's list of virtual pages. If the task is using a shared page group, its reference to the group will be removed, and the group itself will be removed if this was the last task using the shared page group.

Parameters R0 = 9 (opcode)
R1 = first node

Results R0 = error code

Parameter description

First node This is the MEMLST pointer from the task's header.

Error code 0 = No Error. The pages belonging to the task were freed.

>FFFF = Invalid opcode. You attempted to call this from a user task.

GET MEMORY PAGE

Function This routine can not be directly used by tasks under MDOS, it is reserved for use by system library functions.

It is used to get a single memory page, by specific physical page number, or by speed priority.

Parameters R0 = 10 (opcode)
R1 = physical page
R2 = speed flag

Results R0 = error code
R1 = node pointer

Parameter description

Physical page	<p>If this is in the range >00:>FF, MDOS will return a pointer to the memory node for the page, only if the page is currently unassigned.</p> <p>If this is larger than >FF, MDOS will return a pointer to the memory node for the the first free page in the system with the specified speed attribute.</p>
Speed flag	<p>This parameter is used only if the physical page number specified is larger than >FF. If this is zero, MDOS will allocate the first memory page available in the free list. If this is non-zero, MDOS will attempt to allocate the first zero wait state page available from the free list, if there are no zero wait state pages available, MDOS will allocate the first free page it finds.</p>
Error code	<p>0 = No Error. The page was reserved as specified, it is not assigned to any task, and it is not available for use.</p> <p>1 = Page not available. The specified page was not free, or there are no free pages in the entire system.</p> <p>>FFFF = Invalid opcode. You attempted to call this from a user task.</p>
Node pointer	<p>This is pointer to a 4 byte memory node inside of the memory library's address space.</p>

FREE MEMORY PAGE

Function	<p>This routine can not be directly used by tasks under MDOS, it is reserved for use by system library functions.</p> <p>Adds the specified physical page to the list of pages available for use by user tasks.</p>
Parameters	<p>R0 = 11 (opcode)</p> <p>R1 = page number</p>
Results	<p>R0 = error code</p>
Parameter description	
Page number	<p>This a simply a physical page number to be freed.</p>

Error code 0 = No Error. The page was reserved as specified, it is not assigned to any task, and it is not available for use.

8 = Out of table space. MDOS was unable to create a free page because there weren't enough virtual pages nodes available to create a new page in the free pool.

>FFFF = Invalid opcode. You attempted to call this from a user task.

FREE MEMORY NODE

Function This routine can not be directly used by tasks under MDOS, it is reserved for use by system library functions.

This operation will add the specified 4-byte node to the memory nodes available for use by the other memory management library functions.

Parameters R0 = 12 (opcode)
 R1 = node address

Results R0 = error code

Parameter description

Node address This is the address of the 4-byte node within the memory management library's address space.

Error code 0 = No Error. The node was added to the free node list.

>FFFF = Invalid opcode. You attempted to call this from a user task.

LINK MEMORY NODE

Function This routine can not be directly used by tasks under MDOS, it is reserved for use by system library functions.

This is used to link memory nodes together. It can be used to link page nodes onto a task's virtual memory list, to link page nodes to the system free page list, and to link nodes into the free node list.

Parameters

R0	= 13 (opcode)
R1	= new node
R2	= old node

Results R0 = error code

Parameter description

New node The node to be inserted into a list after the old node.

Old node The node, in a node list, after which the new node is to be inserted.

Error code 0 = No Error. The nodes were linked together.

>FFFF = Invalid opcode. You attempted to call this from a user task.

GET MEMORY LIST (system)

Function This routine can not be directly used by tasks under MDOS, it is reserved for use by system library functions.

This routine will return a task's virtual page list to location >1F00 in system page zero. This is used primary by the DSR routines to locate data pointed to by a task's PAB buffer address.

Parameters R0 = 14 (opcode)

Results R0 = -1 (error code)
R0 = page count (no error)

Parameter description

Error code >FFFF = Invalid opcode. You attempted to call this from a user task.

Page count This is the number of valid pages in the page list at >1F00 in system page zero. This count is also returned at >1FFE.