# VMEPROM Version 2/32

## User's Manual

**Revision No. 3**
**March 1997**

P/N 200183
FORCE COMPUTERS Inc./GmbH

# INTRODUCTION

# T A B L E   O F   C O N T E N T S

# 1 General Overview

## 1.1 VMEPROM Modules

VMEPROM is a PDOS based real time Monitor. It consists of two basic parts:

1) PDOS Kernel and File Manager with BIOS modules
2) User Interface

The first part, the PDOS Kernels and the PDOS File Manager, together with the BIOS modules, consume around 32 Kbytes. This part is responsible for all the system calls and the hardware interface. It can be used without the user interface to generate a real time kernel based application package where no user interface is required. The details of the implementation and how customized applications can be generated are described in detail in the User's Manual of your particular CPU board.

The second part is much bigger and contains the complete user interface, the built-in functions and debugging facilities. The size of this part is about 148 Kbytes.

The remaining space in the EPROM is reserved for future expansions.

The VMEPROM package is ported to several FORCE CPU boards. For details on the implementation please refer to the User's documentation of your CPU board. All implementation dependant parts are located in the BIOS modules, which are part of the kernel.

The kernel/file manager features over 100 system calls and is 100% identical to PDOS. Chapter 4 of this manual describes all the details of the available system calls.

The user interface gives the user both a debugging tool and an interface to the system functions and the file manager. It includes breakpoint setting, tracing, a powerful line assembler/disassembler, task management, event control, RAM disk support, disk format and initialization, and a full screen editor.
The size of the complete VMEPROM package is 180 Kbytes. It resides in the on-board EPROM area of the CPU board.

In addition, some boot programs for various operating systems may be put into the EPROMs. For details on layout and use of the EPROMs, please refer to the Introduction to VMEPROM, which is part of the User's Manual of the CPU board.

## 1.2 Features of VMEPROM

### 1.2.1 Debugging Functions

- Line assembler/disassembler with full support of all 68000/68010, 68020/68881, or 68030/68882 instructions.

- Over 20 commands for program debugging, including breakpoints, tracing, processor register display and modify. An optional 68881/68882 floating point coprocessor is also supported (68020/68030 versions only).

- S-record up/downloading from any port in the system.

- Time stamping of user programs.

- Built-in Benchmarks

### 1.2.2 System Functions

- Disk support for RAM disk, floppy and Winchester. Either a SYS68K/WFC-1 or a SYS68K/ISCSI-1 may be used. VMEPROM also allows disk formatting and initialization.

- Serial I/O support for up to two SIO-1/2 or ISIO-1/2 boards in the system.

- EPROM programming utility, using the SYS68K/RR-2/RR-3 boards.

- Full Screen Editor.

- More than 30 commands to control the PDOS kernel and file manager.

- Complete task management.

- I/O redirection on the command line.

### 1.2.3 System Calls

- Over 100 system calls.

- Data conversion functions.

- Task management system calls.

- Terminal I/O functions.

- File management functions.

## 2. History of Manual Revisions

| REVISION | DESCRIPTION | DATE OF LAST CHANGE |
|---|---|---|
| **Ed. 0** | First Revision. | SEP/27/1989 |
| **Ed. 1** | Corrected examples for ’#’ and ’BF’. | FEB/06/1990 |
| **Ed. 2** | Events corrected in *Section 2, Chapter 1.1.5*, **Page 1-12**. | APR/19/1991 |
| **Ed. 3** | Editorial changes | MARCH/1997 |

# VMEPROM SYSTEM OVERVIEW

## TABLE OF CONTENTS



## LIST OF FIGURES

# 1. GENERAL INFORMATION

The VMEPROM kernel and file manager functions are described here in detail. There are four main sections of VMEPROM; namely, the BIOS, kernel, file management module and the user interface.

## 1.1 KERNEL

VMEPROM is based on the powerful PDOS real time kernel.

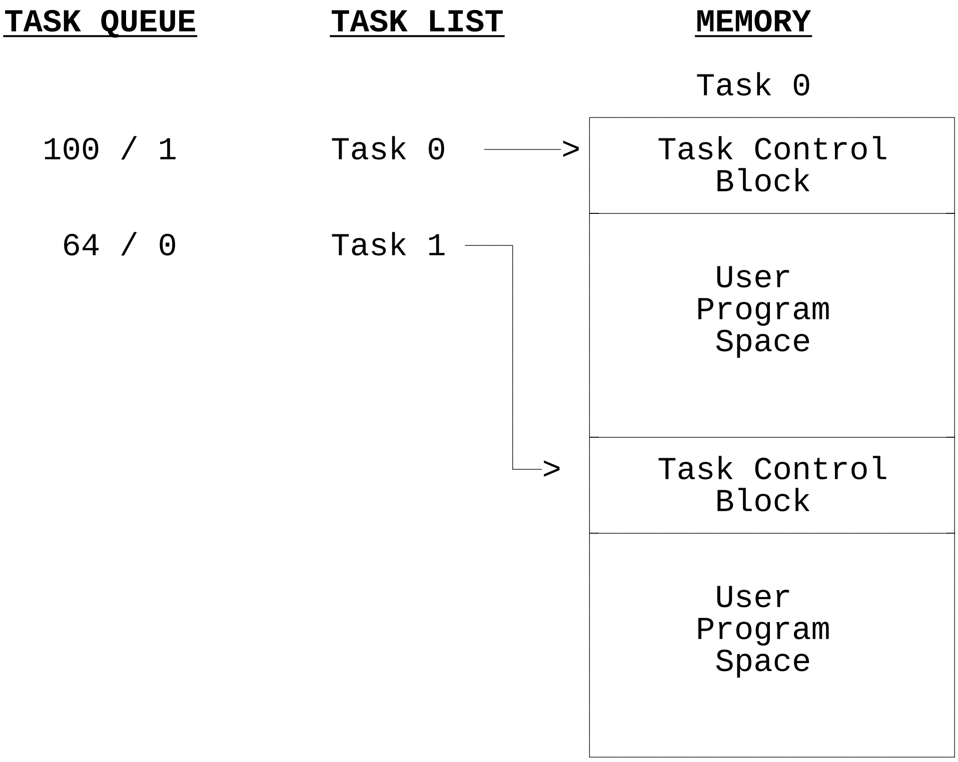
Features of the kernel:

1. Multitasking, multiuser scheduling
2. System clock
3. Memory allocation
4. Task synchronization
5. Task suspension
6. Event processing
7. Character I/O including buffering
8. Support primitives

The PDOS kernel is the multitasking, real time nucleus of the VMEPROM. Tasks are the components comprising a real time application. It is the main responsibility of the kernel to see that each task is provided with the support it requires in order to perform its designated function.

The main responsibilities of the kernel are the allocation of memory and the scheduling of tasks. Each task must share the system processor with other tasks. The kernel saves the task’s context when it is not executing and restores it again when it is scheduled. Other responsibilities of the kernel are maintenance of a 24 hour system clock, task suspension and rescheduling, event processing, character buffering and other support functions.

### 1.1.1 VMEPROM TASK




A task is defined as a program entity which can execute independently of any other program if desired. It is the most basic unit of software within a real time kernel. A user task consists of an entry in the task queue, task list and a task control block with user program space.

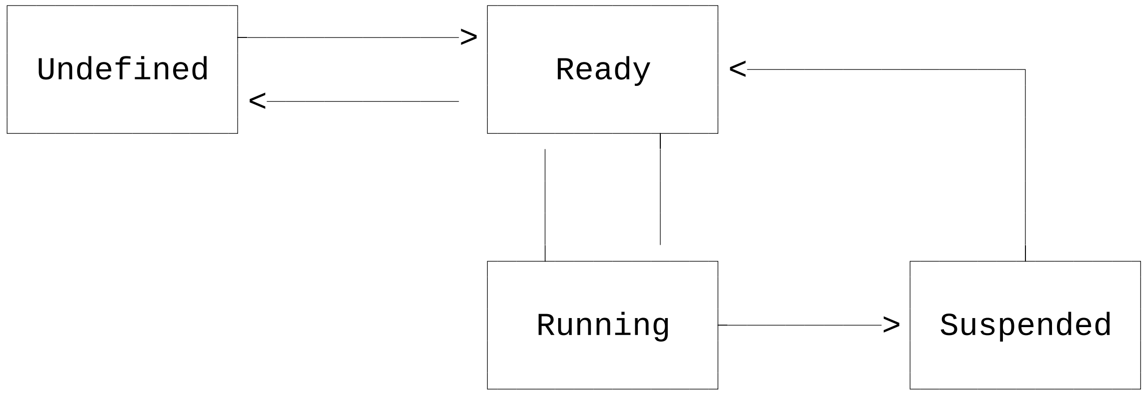
The task queue and list are used by the kernel to schedule tasks. A task queue entry consists of a task priority and a task number. The list is ordered with the highest priority entry first. A task list entry consists of a priority, task time, spawned task number, task control block pointer, task map constant and two suspended event registers. The task number is assigned according to its entry position.

The first $1000 (hex) bytes of a task are the task control block. This block of memory consists of buffers and parameters peculiar to the task. The 680x0 address register A6 points to the status block when the user program is first entered. The task parameters may be referenced by a user program but care must be taken that the kernel is not crashed!

The task control block variables are displacements beyond register A6 and are defined in FIGURE 2.1.

The user program space begins immediately following the task control block. Position independent 680x0 programs are loaded into this area for execution. Task memory is allocated in 2 Kbyte increments with a minimum task size of 8 Kbytes. The total task overhead is $1000 or 4096 bytes.

From the time a task is coded by a programmer until the task is destroyed, it is in one of four task states. Tasks move among these states as they are created, begin execution, are interrupted, wait for events and finally complete their functions. These states are defined as follows:




**1. Undefined**

A task is in this state before it is loaded into the task list. It can be a block of code in a disk file or stored in memory.

**2. Ready**

When a task is loaded in memory and entered in the task queue and task list but not executing or suspended, it is said to be ready.

**3. Running**

A task is running when scheduled by the VMEPROM kernel from the task list.

**4. Suspended**

When a task is stopped pending an event external to the task, it is said to be suspended. A suspended task moves to the ready or running state when the event occurs.

A task remains undefined until it is made known to the operating system by making an entry in the task queue. Once entered, a task immediately moves to the ready state which indicates that it is ready for execution. When the task is selected for execution by the scheduler, it moves to the run state. It remains in the run state until the scheduler selects another task or the task requires external information and suspends itself until the information is available. The suspended state greatly enhances overall system performance.

### 1.1.2 MULTITASKING

VMEPROM allows 64 independent tasks to reside in memory and share CPU cycles. Each task contains its own task control block and thus executes independently of any other task. A task control block consists of buffers, pointers and stack areas. Four parameters are required for any new task generation. These are:

1) **A task priority.** The range is from 255 (highest priority) to 1 (lowest priority).

2) **A task time.** It ranges from 1 (1*10ms) to 255 (255*10ms).

3) **Tasking memory.** Memory is allocated to a task in 2 Kbyte increments. The minimum task size is 8 Kbytes and the first $1000 or 4096 bytes are assigned to the task TCB.

4) **An I/O port.** Input ports are unique while many tasks may share the same output port for task console communication. Port 0 is used as a phantom port if no I/O is required for the task.

5) **A task command or memory address.**

Each of the above requirements defaults to a system parameter. Task priority defaults to the parent task's priority minus one. Task time defaults to 1.

The default memory allocation is 8 Kbytes and the default console port is the phantom port. The default command is the VMEPROM user interface.

However, if the VMEPROM Monitor is the task to be started and no input is possible then the new task can receive commands to execute via a task message.

A task entry in the task list consists of a task number designation, parent task number, time interval, task priority, memory map constant, task control block pointer and two event registers. Swapping from one task to the next is done when the task interval timer decrements to zero, during an I/O system call or when an external event causes a context switch. The task interval timer decrements by one every ten milliseconds.

Any task may spawn another task. Memory for the new task is allocated in 2 Kbyte blocks from a pool of available memory. If no memory is free, the spawning task's own memory is used and the parent task's memory is reduced in size by the amount of memory allocated to the new task. It is important to note that some assembly coded programs and all high level language programs use both the low and high addresses of the task memory. To prevent memory loss from a task and program failure, it is necessary to allocate enough memory to the free memory pool before creating a new task under program control. Otherwise, the task may give up its variable space or stack to the spawned task.

The VMEPROM kernel maintains a memory bit map to indicate which segments of memory are currently in use. Allocation and deallocation are in 2 Kbyte increments. When a task is terminated, the task’s memory is automatically deallocated in the memory bit map and made available for use by other tasks.

"Multiuser" refers to spawning new tasks for additional operators. Each new task executes programs or even spawns additional tasks. Such tasks are generated or terminated as needed. Task 0 is referred to as the system task and cannot be terminated.

Figure 2.1 shows the task control block.

**WARNING: Although the locations of the task control block are made available to the user, you must be cautious when using these locations. Many system calls use these locations to perform their functions and any location may change at any time as a result of these system calls. The TCB format may be changed with future versions of VMEPROM.**

**Figure 2.1: Task Status Control Definitions**

**The following are Task Control Block (TCB) definitions:**

```
#define MAXARG          10      /* max argument count of the cmd line   */
#define MAXBP           10      /* max 10 breakpoints                   */
#define MAXNAME         5       /* max 5 names in name buffer           */
#define TMAX            64      /* Max number of tasks                  */
#define ARGLEN          20      /* maximum argument length              */

/* special system flags for VMEPROM */

#define SOMEREG 0x0001  /* display only PC,A7,A6,A5                     */
#define T_DISP  0x0002  /* no register display during trace (TC > 1)    */
#define T_SUB   0x0004  /* trace over subroutine set                    */
#define T_ASUB  0x0008  /* trace over subroutine active                 */
#define T_RANG  0x0010  /* trace over range set                         */
#define REG_INI 0x0020  /* no register initialization if set            */
#define RE_DIR  0x0040  /* output redirection into file and console at  */
                        /* the same time                                */

/* the 68020 regs are stored in the following order:                    */

#define VBR     0
#define SFC     1
#define DFC     2
#define CAAR    3
#define CACR    4
#define PC      5
#define SR      6
#define USTACK  7
#define SSTACK  8
#define MSTACK  9
#define D0      10              /*  10-17  =  D0-D7                     */
#define A0      18              /*  18-24  =  A0-A6                     */

#define N_REGS  25

#define BYTE    unsigned char
#define WORD    unsigned int
#define LWORD   unsigned long

struct TCB{
/*000*/ char _ubuf[256];        /* 256 byte user buffer                */
/*100*/ char _clb[80];          /* 80 byte monitor command line buffer */
/*150*/ char _mwb[32];          /* 32 byte monitor parameter buffer    */
/*170*/ char _mpb[60];          /* monitor parameter buffer            */
/*1AC*/ char _cob[8];           /* character out buffer                */
/*1B4*/ char _swb[508];         /* system work buffer/task pdos stack  */
/*3B0*/ char *_tsp;             /* task stack pointer                  */
/*3B4*/ char *_kil;             /* kill self pointer                   */
/*3B8*/ long _sfp;              /* RESERVED FOR INTERNAL PDOS USE      */
/*3BC*/ char _svf;              /* save flag -- 68881 support (x881)   */
/*3BD*/ char _iff;              /* RESERVED FOR INTERNAL PDOS USE      */
/*3BE*/ long _trp[16];          /* user TRAP vectors                   */
/*3FE*/ long _zdv;              /* zero divide trap                    */
/*402*/ long _chk;              /* CHCK instruction trap               */
/*406*/ long _trv;              /* TRAPV Instruction trap              */
/*40A*/ long _trc;              /* trace vector                        */
/*40E*/ long _fpa[2];           /* floating point accumulator          */
```

**Figure 2.1: Task Status Control Definitions (cont'd)**

```
/*416*/ long *_fpe;              /* fp error processor address          */
/*41A*/ char *_clp;              /* command line pointer                */
/*41E*/ char *_bum;              /* beginning of user memory            */
/*422*/ char *_eum;              /* end user memory                     */
/*426*/ char *_ead;              /* entry address                       */
/*42A*/ char *_imp;              /* internal memory pointer             */
/*42E*/ int  _aci;               /* assigned input file ID              */
/*430*/ int  _aci2;              /* assigned input file ID's            */
/*432*/ int  _len;               /* last error number                   */
/*434*/ int  _sfi;               /* spool file id                       */
/*436*/ BYTE _flg;               /* task flags (bit 8=command line echo)*/
/*437*/ BYTE _slv;               /* directory level                     */
/*438*/ char _fec;               /* file expansion count                */
/*439*/ char  _spare1;           /* reserved for future use             */
/*43A*/ char _csc[2];            /* clear screen characters             */
/*43C*/ char _psc[2];            /* position cursor characters          */
/*43E*/ char _sds[3];            /* alternate system disks              */
/*441*/ BYTE _sdk;               /* system disk                         */
/*442*/ char *_ext;              /* XEXT address                        */
/*446*/ char *_err;              /* XERR address                        */
/*44A*/ char _cmd;               /* command line delimiter              */
/*44B*/ BYTE _tid;               /* task id                             */
/*44C*/ char _ecf;               /* echo flag                           */
/*44D*/ char _cnt;               /* output column counter               */
/*44E*/ char _mmf;               /* memory modified flag                */
/*44F*/ char _prt;               /* input port #                        */
/*450*/ char _spu;               /* spooling unit mask                  */
/*451*/ BYTE _unt;               /* output unit mask                    */
/*452*/ char _u1p;               /* unit 1 port #                       */
/*453*/ char _u2p;               /* unit 2 port #                       */
/*454*/ char _u4p;               /* unit 4 port #                       */
/*455*/ char _u8p;               /* unit 8 port #                       */
/*456*/ char _spare2[26];        /* reserved for system use             */

/***********************************************************************/
/*      VMEPROM variable area                                          */
/***********************************************************************/

/*470*/ char  linebuf[82];       /* command line buffer                  */
/*4C2*/ char  alinebuf[82];      /* alternate line buffer                */
/*514*/ char  cmdline[82];       /* alternate cmdline for XGNP           */
/*566*/ int   allargs, gotargs;  /* argc save and count for XGNP         */
/*56A*/ int   argc;              /* argument counter                     */
/*56C*/ char  *argv[MAXARG];     /* pointer to arguments of the cmd line */
/*594*/ char  *odir, *idir;      /* I/O redirection args from cmd line   */
/*59C*/ int   iport,oport;       /* I/O port assignments                 */
/*5A0*/ char  *ladr;             /* holds pointer to line in _mwb        */
/*5A4*/ LWORD offset;            /* base memory pointer                  */
/*5A8*/ int   bpcnt;             /* num of defined breakpoints           */
/*5AA*/ LWORD bpadr[MAXBP];      /* breakpoint address                   */
/*5D2*/ WORD  bpinst[MAXBP];     /* breakpoint instruction               */
/*5E6*/ char  bpcmd[MAXBP][11];  /* breakpoint command                   */
/*654*/ WORD  bpocc[MAXBP];      /* # of times the breakpoint should be  */
                                 /* skipped                              */
/*668*/ WORD  bpcocc[MAXBP];     /* # of times the breakpoint is already */
                                 /* skipped                              */
/*67C*/ LWORD bptadr;            /* temp. breakpoint address             */
```

**Figure 2.1: <u>Task Status Control Definitions (cont'd)</u>**

```
/*680*/ WORD  bptinst;          /* temp. breakpoint instruction         */
/*682*/ WORD  bptocc;           /* # of times the temp. breakpoint      */
                                /* should be skipped                    */
/*684*/ WORD  bptcocc;          /* # of times the temp. breakpoint is   */
                                /* already skipped                      */
/*686*/ char  bptcmd[11];       /* temp. breakpoint command             */
/*691*/ char  outflag;          /* output messages (yes - 1,no - 0)     */
/*692*/ char  namebn[MAXNAME][8];  /* Name buffer, name                 */
/*6BA*/ char  namebd[MAXNAME][40]; /* Name buffer, data                 */
/*782*/ WORD  errcnt;           /* error counter for test ..            */
/*784*/ LWORD times,timee;      /* start/end time                       */
/*78C*/ LWORD pregs[N_REGS];    /* storage area of processor regs       */
/*7F0*/ WORD  tflag;            /* trace active flag                    */
/*7F2*/ WORD  tcount;           /* trace count                          */
/*7F4*/ WORD  tacount;          /* active trace count                   */
/*7F6*/ WORD  bpact;            /* break point active flag              */
/*7F8*/ LWORD savesp;           /* save VMEprom stack during GO/T etc   */
/*7FC*/ char  VMEMSP[202];      /* Master stack, handle w/ care         */
/*8C6*/ char  VMESSP[802];      /* supervisor stack, handle w/ care     */
/*BE8*/ char  VMEPUSP[802];     /* vmeprom internal user stack          */
/*F0A*/ LWORD f_fpreg[3*8];     /* floating point data regs             */
/*F6A*/ LWORD f_fpcr;           /* FPCR reg                             */
/*F6E*/ LWORD f_fpsr;           /* FPSR reg                             */
/*F72*/ LWORD f_fpiar;          /* FPIAR reg                            */
/*F76*/ BYTE  f_save[0x3c];     /* FPSAVE for null and idle             */
/*FB2*/ BYTE  cleos[2];         /* clear to end of screen parameter     */
/*FB4*/ BYTE  cleol[2];         /* clear to end of line parameters      */
/*FB6*/ char  u_prompt[10];     /* user defined prompt sign             */
/*FC0*/ long  c_save;           /* save Cache control register          */
/*FC4*/ long  exe_cnt;          /* execution count                      */
/*FC8*/ BYTE  nokill;           /* kill task with no input port         */
/*FC9*/ BYTE  u_mask;           /* unit mask for echo                   */
/*FCA*/ WORD  sysflg;           /* system flags used by VMEPROM         */
                                /*   bit 0: display registers short form*/
                                /*   bit 1: trace without reg. display  */
                                /*   bit 2: trace over subroutine       */
                                /*   bit 3: trace over subroutine active*/
                                /*   bit 4: trace over range            */
                                /*   bit 5: no register initialization  */
                                /*   bit 6: output redirection into     */
                                /*          file and console at the     */
                                /*          same time                   */
/*FCC*/ LWORD t_range[2];       /* start/stop PC for trace over range   */
/*FD4*/ LWORD ex_regs;          /* pointer to area for saved regs       */
/*FD8*/ BYTE  sparend[0x1000-0xFD8]; /* make tcb size $1000 bytes       */
        char _tbe[0];           /* task beginning                       */
   };
```

### 1.1.3 SYSTEM SERVICES

System services are those functions that a task requires of the operating system while entered in the task list. These requirements range from timing and interrupt handling to task coordination and resource allocation.

VMEPROM provides many time-oriented functions which key off of the system hardware interval timer. The current time of day and the date are maintained with fine adjustment parameters. A 32 bit counter is used for various delta time functions such as task scheduling and event delays.

Hardware interrupts are processed by the kernel BIOS or passed to user tasks. Tasks can be suspended pending the occurrence of an interrupt and then be rescheduled when the interrupt occurs. Interrupts such as the interval timer and character input or output are handled by the kernel itself.

Task coordination is an integral part of real time applications since many functions are too large or complex for any single task. The kernel uses common or shared data areas, called mailboxes, along with a table of preassigned bit variables, called events, to synchronize tasks. A task can place a message in a mailbox and suspend itself on an event waiting for a reply. The destination task is signaled by the event, looks in the mailbox, responds through the mailbox and resets the event signaling the reply.

System resources include the processor itself, system memory and support peripherals. The kernel provides primitives to create and delete tasks from the task list. Memory is allocated and deallocated as required. Peripherals are generally a function of the file manager but are assigned and released via system events. Device drivers coordinate related I/O functions, interrupts and error conditions. All of these functions are available to user tasks and thus tasks may spawn tasks and dynamically control their operating environment.

Other support utilities contained within the kernel of VMEPROM include number conversion, command line decoding, date and time conversions and message processing routines. Facilities are also provided for locking a task in the run state during critical code execution.

### 1.1.4 CHARACTER I/O

The flow of character data through the kernel is the most visible function of VMEPROM. Character buffering or type-ahead assures the user that each keyboard entry is logged, even when the application is not looking for characters. Character output is through program control (polled I/O).

Inputs and outputs are through logical port numbers. A logical port is bound to a physical UART (Universal Asynchronous Receiver/Transmitter) by the baud port commands. Only one task is assigned to an input port at any one time while many tasks may share the same output port. It is then the responsibility of each task to coordinate all outputs.

Character inputs come from four sources:

1. User memory
2. Disk file
3. I/O device driver
4. System input port buffer

The source is dictated by input variables within the task control block. Input variables are the Input Message Pointer (IMP$(A6)), Assigned Console Input (ACI$(A6)) and input port number (PRT$(A6)).

When a request is made by a task for a character and IMP$(A6) is nonzero, then a character is retrieved from the memory location pointed to by IMP$(A6). IMP$(A6) is incremented after each character. This continues until a null byte is encountered, at which time IMP$(A6) is set to zero.

If IMP$(A6) is zero and ACI$(A6) is nonzero, then a request is made to the file manager to read one character from the file assigned to ACI$(A6). The character then comes from a disk file or an I/O device driver. This continues until an error occurs (such as an END-OF-FILE) at which time the file is closed and ACI$(A6) is cleared.

If both IMP$(A6) and ACI$(A6) are zero, then the logical input port buffer selected by PRT$(A6), is checked for a character. If the buffer is empty, then the task is automatically suspended until a character interrupt occurs.

VMEPROM character outputs are directed to various destinations according to output variables in the task control block. Output variables are the output unit (UNT$(A6)), spooling unit (SPU$(A6)), spooling file ID (SFI$(A6)) and output port variables U1P$(A6), U2P$(A6), U4P$(A6) and U8P$(A6). The output unit selects the different destinations. (This is NOT to be confused with disk unit numbers.)

When an output primitive is called, the task output unit is ANDed with the task spooling output unit. If the result is nonzero, then the character is directed to the file manager and written to the file specified by SFI$(A6). The output unit is then masked with the complement of the spooling unit and passed to the UART character output processor.

Units 1, 2, 4 and 8 are special output numbers. Unit 1 is the console output port assigned when the task was created. Units 2, 4 and 8 are optional output ports that correspond to TCB variables U2P$, U4P$ and U8P$. They are assigned by the command ASSIGN or baud port command.

If the 1 bit (LSB) is set in the masked output unit (UNT$(A6)), then the character is directed to port U1P$(A6). Likewise, if bits 2, 3 or 4 are set in the masked output unit, then the character is output to the U2P$(A6), U4P$(A6) or U8P$(A6) ports.

In summary, the bit positions of the output unit are used to direct output to various destinations. More than one destination can be specified. Bits 1 through 4 are predefined according to U1P$, U2P$, U4P$ and U8P$ variables within the task control block. Other unit bits are used for outputs to files and device drivers. Thus, if SPU$(A6)=4 and UNT$(A6)=7, then output would be directed to the file manager via SFI$(A6) and to two UARTs as specified in U1P$(A6) and U2P$(A6).

```
SPU$(A6) = 0000 0000 0000 0100
UNT$(A6) = 0000 0000 0000 0111
                             |||
       File SFI$(A6) --------+||
       Port U2P$(A6) ---------+|
       Port U1P$(A6) ----------+
```

### 1.1.5 EVENTS

Tasks communicate by exchanging data through mailboxes. Tasks synchronize with each other through events. Events are single bit flags that are global to all tasks.

4 types of event flags:

```
   1-63 = Software
  64-80 = Software resetting
 81-127 = System
    128 = Local to task
```

There are four types of event flags: Software, Software Resetting, System and Local. System events are further divided into output, input, timing, driver and system resource events. System events are predefined software resetting events that are set during VMEPROM initialization. Event 128 is local to each task and is used as a delay event.

1) Events 1 through 63 are software events. They are set and reset by tasks and not changed by the task scheduling. A task can suspend itself pending a software event and then be rescheduled when the event is set. One task must take the responsibility of resetting the event for the sequence to occur again.

2) Events 64 through 80 are like the normal software events except that the kernel automatically resets the event whenever a task suspended on that event is rescheduled. Thus, one and only one task is rescheduled when the event occurs.

   These events are set and reset by the Send Message Pointer (XSMP) and Get Message Pointer (XGMP) primitives.

3) Events 81 through 95 are reserved for future use by VMEPROM.

4) Events 96 through 111 correspond to input ports 0 through 15. A task suspends itself on an input event if a request is made for a character and the buffer is empty. Whenever a character comes into an interrupt

driven input port buffer, the corresponding event is set.

5) Events 112 through 115 are timing events and are set automatically by the clock module according to intervals defined in the Basic I/O module (BIOS).

```
112 = 1/5 second event
113 =  1  second event
114 = 10  second event
115 = 20  second event
```

A task suspended on one of these events is regularly scheduled on a tic or second boundary.

6) Events 116 through 127 are for system resource allocation. Drivers and other utilities requiring ownership of a system resource synchronize on these events. These events are initially set by the kernel, indicating the resource is available. One and only one task at a time is allowed access to the resource. When the task is finished with the resource, it must reset the event thus allowing other tasks to gain access.

7) Event 128 is local to each task. Unlike other events, it can only be set by a delay primitive (XDEV). It is automatically reset by the scheduling of a task suspended on event 128.

Many different methods are available for intertask communication. Most involve a mailbox technique where semaphores are used to control message traffic. Specially designed memory areas such as MAIL, COM and event flags allow high level program communications. VMEPROM maintains 64 message buffers for queued message communications between tasks or console terminals. More sophisticated methods require program arbitrators and message buffers.

The MAIL array is a permanent 254 byte memory buffer accessible by assembly language programs. Its address is located in the second long word of SYRAM {4(A5)}.

**Event flags:**

Event flags are global system memory bits, common to all tasks. They are used in connection with task suspension or other mailbox functions. Events 1 through 63 are for software communication flags. Events 64 through 127 automatically reset when a suspended task is rescheduled. Events 81 through 95 are output events; 96 through 111 are input events; 112 through 115 are timing events; and 116 through 127 are system events. Event 128 is local to each task and cannot be used to communicate between tasks.

**Message buffers:**

VMEPROM maintains 64  64 byte message buffers for intertask communication. A message consists of up to 64 bytes plus a destination task number. More than one message may be sent to any task.

**Message pointers:**

VMEPROM supports shorter message pointer transfers between tasks with the Send Message Pointer (XSMP) and Get Message Pointer (XGMP) primitives. When a pointer is sent, event [destination message slot # + 64] is set. When a message pointer is retrieved, the corresponding event is cleared. These messages are not queued, but are much faster for intertask message passing than the queued 64 byte messages.

**Memory Mailbox:**

The FM monitor command is used to permanently allocate system memory for nontasking data or program storage. Memory allocated in this way can be used for mailbox buffers as well as handshaking semaphores or assembly programs.

### 1.1.6 TASK SUSPENSION

Any task can be suspended pending one or two events. Software events (1-127) are system memory bits global to all tasks. Event 128 is local to each task. A suspended task does not receive any CPU cycles until one of the desired events occurs. A task is suspended from an assembly language program by the XSUI primitive. A suspended task is indicated in the LIST TASK (LT) command by the event number(s) being listed under the ’Event’ heading.

When one of the events occurs, the task is rescheduled and resumes execution. If the event is set by the XSEF primitive, then an immediate context switch occurs. If a high priority task is waiting for the event, it is immediately rescheduled, overriding any current task (unless locked). If the event is set with a XSEV primitive, then the task begins execution during the normal swapping function of the kernel.

### 1.1.7 HIGH PRIORITY TASKS

A high priority task is defined as a task in the execution list which is exempt from round robin scheduling. This means the task will continue to execute until it suspends itself (due to I/O or if an XSUI command is executed,) or a higher priority task becomes ready. Task priority is listed by the LT (List Task) command. A task priority can be altered with the ’TP’ command.

High priority tasks are useful in writing user interrupt handlers where immediate and fast response is required.

## 1.2 FILE MANAGEMENT

The file management module of VMEPROM supports sequential, random, read only and shared access to named files on a secondary storage device. These low overhead file primitives use a linked, random access file structure and a logical sector bit map for allocation of secondary storage. No file compaction is ever required. Files are time stamped with date of creation and last update. VMEPROM allows up to 64 files to be open simultaneously.

### 1.2.1 FILE STORAGE

A file is a named string of characters on a secondary storage device. A group of file names is associated together in a file directory. File directories are referenced by a disk number. This number is logically associated with a physical secondary storage device by the read/write sector primitives. All data transfers to and from a disk number are blocked into 256 byte records called sectors.

A file directory entry contains the file name, directory level, the number of sectors allocated, the number of bytes used, a start sector number and dates of creation and last update. A file is opened for sequential, random, shared random or read only access. A file type of 'DR' designates the file to be a system I/O driver. A driver consists of up to 252 bytes of position independent binary code. It is loaded into the channel buffer whenever opened. The buffer then becomes an assembly program that is executed when referenced by I/O calls. (The driver facility is described in detail in the Appendix).

A sector bit map is maintained for each disk number. Associated with each sector on the logical disk is a bit which indicates if the sector is allocated or free. Using this bit map, the file manager allocates (sets to 1) and deallocates (sets to 0) sectors when creating, expanding and deleting files. Bad sectors are permanently allocated. When a file is first defined, one sector is initially allocated to that file and hence, the minimum file size is one sector.

A file is accessed through an I/O channel called a file slot. Each file slot consists of a 38 byte status area and an associated 256 byte sector buffer. Data movement is always to and from the sector buffer according to a file pointer maintained in the status area. Any reference to data outside the sector buffer requires the buffer to be written to the disk (if it was altered) and the new sector to be read into the buffer. The file manager maintains current file information in the file slot status area such as the file pointer, current sector in memory, END-OF-FILE sector number, buffer in memory flag and other critical disk parameters required for program-file interaction.

Eight sector buffers are actually memory resident at a time. The file manager allocates these buffers to the most recently accessed file slots. Every time a file slot accesses data within its sector buffer, the file manager checks to see if the sector is currently in memory. If it is, the file slot number is rolled to the top of the most recently accessed queue. If the buffer has been previously rolled out to disk, then the most recently accessed queue is rolled down and the new file slot number is placed on top. The file slot number rolled out the bottom references the fourth last accessed buffer which is then written out to the disk. The resulting free buffer is then allocated to the calling file slot and the former data restored.

Files requiring frequent access generally have faster access times than those files which are seldom accessed. However, all file slots have regular access to buffer data.

File storage on disk is allocated in sector increments. All sectors are both forward and backward linked. This facilitates the allocation and deallocation of sectors as well as random or sequential movement through the file.

All files are accessed in either sequential or random access mode. Essentially, the only difference between the two modes is how the End-Of-File pointers are handled when the file is closed. If a file has been altered, sequential mode updates the EOF pointer in the disk file directory according to the current file byte pointer, whereas the random mode only updates the EOF pointer if the file has been extended.

Two additional variations of the random access mode allow for shared file and read only file access. A file which has been opened for shared access can be referenced by two or more different tasks at the same time. Only one file slot and one file pointer are used no matter how many tasks open the file. Hence, it is the responsibility of each user task to ensure data integrity by using the lock file or lock process commands. The file must be closed by all tasks when the processing is completed.

A read only random access to a file is independent of any other access to that file. A new file slot is always allocated when the file is read only opened and a write to the file is not permitted.

### 1.2.2 FILE NAMES

Example for Legal file names:

```
FILE
A1234567:890;255/127
PROGRAM/3
FILE2;10
```

VMEPROM file names consist of an alphabetic character (A-Z or a-z) followed by up to seven additional characters. An optional one to three character extension is separated from the file name by a colon (:). Other optional parameters include a semi-colon (;) followed by a file directory level and a slash (/) followed by a disk number. The file directory level is a number ranging from 0 to 255. The disk number ranges from 0 to 255.

A file typed as a system I/O device driver has entry points directly into the channel buffer for OPEN, CLOSE, READ, WRITE and POSITION commands.

If the file name is preceded by a '#', the file is created (if undefined) on all open commands except for read only open. When passing a file name to a system primitive, the character string begins on a byte boundary and is terminated with a null.

Special characters such as a period or a space may be used in file names. However, such characters may restrict their access. The command line interpreter uses spaces and periods for parsing a command line.

### 1.2.3 DIRECTORY LEVELS

Each disk directory is partitioned into 255 directory levels. Each file resides on a specific level, which facilitates selected directory listings. You might put system commands on level 0, procedure files on level 1, object files on level 10, listing files on level 11 and source files on level 20. Level 255 is global and references all levels.

A current directory level is maintained and used as the default level in defining a file or listing the directory when no directory level is specified.

### 1.2.4 DISK NUMBERS

A disk number is used to reference a physical secondary storage device and facilitates hardware independence. All data transfers to and from a disk are blocked into 256 byte records called sectors.

The range of disk numbers is from 0 to 255. Several disk numbers may share the same secondary storage device. Each disk can have a maximum of 65280 sectors or 16,711,680 bytes.
A default disk number is assigned to each executing task and stored in the task control block. This disk number is referred to as the system disk and any file name which does not specifically reference a disk number defaults to this parameter.

Up to four disk devices can be associated with each task. When a file is referenced, each directory is searched (in order) until the file is found.

When a task is created, the parent task's disk number(s) and directory level are copied into the task control block of the new task.

### 1.2.5 FILE ATTRIBUTES

Associated with each file is a file attribute. File attributes consist of a file type, storage method and protection flags. These parameters are maintained in the file directory and used by the VMEPROM monitor and file manager.

The following are legal VMEPROM file types:

1. **Batch processes:** AC (Assign console). A file typed 'AC' specifies to the VMEPROM monitor that all subsequent requests for console character inputs are intercepted and the character obtained from the assigned file.

2. **System file:** SY. A 'SY' file is a directly loadable 680x0 hex file. The entry address is the first address within the file.

3. **ASCII text file:** TX (ASCII text file). A 'TX' type classifies a file as one containing ASCII character text.

4. **I/O driver:** DR. A 'DR' file type indicates that the file data is an I/O driver program and is executed when referenced. An I/O driver file cannot be copied with the CF command.

The following file attributes which are supported by the PDOS
operating system are not supported by VMEPROM:

Binary file: BN
PDOS Tagged object: OB
Basic binary and source: BX and EX

Files are physically stored in contiguous sectors whenever possible. A non-contiguous structure results from file expansions where no contiguous sectors are available. Contiguous files have random access times far superior to non-contiguous files. A contiguous file is indicated in the directory listing by the letter 'C' following the file type. File protection flags determine which commands are legal when accessing the file. A file can be delete- and/or write protected.

File storage method and protection flags are summarized as follows:

1) **Contiguous file: C**

   A contiguous file is organized on the disk with all sectors logically and sequentially in order. Random access in a contiguous file is much faster than in a non-contiguous file since the forward/backward links are not required for positioning.

2) **Delete protect: ***

   A file which has one asterisk as an attribute cannot be deleted from the disk until the attribute is changed.

3) **Delete and write protect: ****

   A file which has two asterisks as an attribute cannot be deleted nor written to. Hence, READ, POSITION, REWIND, OPEN and CLOSE are the only legal file operations.

4) **File altered: +**

   A file which has a plus sign as an attribute has been altered.

### 1.2.6 TIME STAMPING

Whenever VMEPROM is cold-started after power up or RESET, the time and date should first be initialized with the ID command. These values are then maintained by the system clock and are used for time stamping file updates and other user defined functions.

When a file is first created or defined, the current date and time is stored with the disk directory entry. This time stamping appears in the 'DATE CREATED' section of a directory listing. From then on, the creation date and time are not changed.

When a file has been opened, altered and then closed, the current date and time are written to the 'LAST UPDATE' section of the disk directory entry. The time stamp indicates when the file was last altered by any user.

### 1.2.7 PORTS, UNITS and DISKS

The terms ports, units and disks are often confused and hence are explained again:

**Ports**

Ports are logical input channels and are referenced by numbers 0 through 15. Associated with each port is an interrupt driven input buffer. The BAUD PORT (BP) command binds a physical UART to a buffer and initializes the chip with baud rate and character format.

**Units**

A unit is an output gating variable. Each bit of the variable directs character output to a different source. Bit 1 (LSB) is associated with U1P$(A6) output port. Likewise, bit 2 is associated with U2P$(A6) output port. The 'ASSIGN' command can be used to bind different output ports to a output channel.

**Disks**

A disk is a logical reference to a secondary storage device. Disk numbers range from 0 to 255.

# VMEPROM BUILT-IN COMMANDS

## TABLE OF CONTENTS

## TABLE OF CONTENTS (cont'd)

## 1.0 GENERAL INFORMATION

The VMEPROM command interpreter is a set of resident routines for program debugging and handling of the most common kernel functions.

The command interpreter searches for a given command in the following sequence:

1. Is the command defined in the name table ?
2. Is it a built-in command ?
3. Is the command available as an installed utility ?
4. Is the command available as a disk file on the current system disk ?

If a match is found in any of the above steps, the command is executed.

The prompt of VMEPROM is a single question mark, followed by a space ("? ").

## 1.1 Command Line Syntax and Line Editing

### 1.1.1 Command Line Arguments

The VMEPROM command interpreter allows several options. In general the complete command line is divided into separate arguments. The arguments must be separated by one or more spaces or a comma. If a null-argument has to be entered, it must be represented by a comma only.

```
Example: ? PROG ARG1,,ARG3,
```

In this example, the arguments number 2 and 4 are null-arguments.

If any argument is using a comma, space, period or one of the I/O redirection arrows, it has to be put in brackets to suspend the command line interpretation.

```
Example: ? PROG1 (Hello, world.),(<....>),>TEMP
```

The file TEMP now contains the output of PROG1 which may be:

```
? SF TEMP
ARGUMENT 1 was: Hello, world.
ARGUMENT 2 was: <....>
ARGUMENT 3 was:
ARGUMENT 4 was:
ARGUMENT 5 was:

?
```

### 1.1.2 <u>Input/Output Redirection</u>

VMEPROM supports simple I/O redirection. The specifiers are the signs '<' for input and '>' or '>>' for output and may appear at any location in the command line, but must be after the command name. Immediately after the redirection signs '<' and '>', a port number or a filename must be specified. After the sign '>>' only a valid filename should be given. The redirection signs '<' and '>' stand for outputting data only to the specified port or file. The sign '>>' declares the output as written to a file additionally.

The port number may be one of the ports available in the system. It is expected to be given in hexadecimal number system.

The filenames for I/O redirection may be any file. If it does not exist, it will be created.

**NOTE:** Incorrect port numbers or filenames can lead to unforeseen outputs.

The arguments specifying the I/O redirection are removed from the command line by the command interpreter and do not appear in the user program or the built-in command.

**Example: ? PROG <TEMP >3 ARG1,ARG2,ARG3,ARG4**

In this example, the program PROG is started. It is getting all inputs from the TEMP and all output is redirected to port 3.

The I/O redirection uses the following PDOS functions:

- Input from file uses the assigned console input file mechanism of PDOS.

- Input from port reassigns the input port number (PRT$) in the TCB temporarily.

- Output to file uses the spool file mechanism of PDOS together with the Unit 4 port. So the Unit 4 port shall not be used.

- Output to port reassigns the output port number (U1P$) in the TCB temporarily.

### 1.1.3 Multiple Commands

VMEPROM allows command lines of up to 78 characters. This command line can contain several different commands. The parsing of the command line is terminated at the first period (".") and the remaining command line is saved to be used later.

```
     Example: ? RM D0 12345678.SM 2,Hello
              ? SM 2,Hello
              ?
```

Be careful when modifying a floating point register from the command line as the decimal point is interpreted as a command line separator. If a floating point register has to be modified, the number must be put in brackets.

```
     Example: ? RM FP0 (12.345).SM 2,Hello
              ? SM 2,Hello
              ?
```

### 1.1.4 Command Line Editing

The PDOS get line (XGLM) primitive is used to get a command line of up to 78 characters into the command line buffer.

Input is normally in replace mode which means an incoming character replaces the character at the cursor. Various control characters can be used to edit the input line.

The following table summarizes the control characters:

```
            [ESC] = Cancel current line
         [CTRL-C] = Cancel current line
         [CTRL-I] = Enter insert mode
         [CTRL-A] = Recall last entered line
         [CTRL-L] = Move right 1 character
         [CTRL-H] = Move left 1 character
         [CTRL-D] = Delete character under cursor
         [RUBOUT] = Delete 1 character to the left
```

A [CTRL-I] changes input from replace to insert mode. The mode returns to replace mode when any other editing control code is entered. Replace mode overwrites the character under the cursor. Insert mode inserts a character at the current cursor position.

In either mode, the cursor need not be at the end of the line when the [CR] is entered. The command line is passed as it appears on the screen.

When a line is accepted, it is copied to another buffer (MPB$) where it can be recalled by using the [CTRL-A] character. A [CR] and [LF] are output to the console followed by the recalled line. The cursor is positioned at the end of the line. This is a circular buffer and commands will rotate through it as they are recalled.

Numeric parameters are entered as signed decimal, unsigned hex, unsigned octal or unsigned binary numbers. All numbers are converted to two's complement 32-bit or 16-bit integers depending on their function. Therefore it ranges from -2,147,483,648 to 2,147,483,647 (hex $80000000 to $7FFFFFFF) or -32,768 to 32,767 (hex $8000 to $7FFF). All built-in commands assume that numbers are entered in hex if not noted otherwise.

To change from the expected number system, numbers must be preceded with a special sign. These are: a dollar sign ($) to enter into hexadecimal, an ampersand (&) to enter into decimal, an at/around sign (@) to enter into octal and a percent sign (%) to enter into the binary number system.

(Note: Numbers are not checked for overflow. Hence, $FFFFFFFF or 4,294,967,295 are equivalent to -1). A line beginning with an '*' is ignored. This is very useful to insert comment lines in command files.

### 1.1.5 Line Editing

Some commands allow inputting data outside the command line. For this a line editor is used. There are some control characters to edit the line:

```
   [ESC] = Cancel current line and exit
[CTRL-C] = Cancel current line and exit
[CTRL-I] = Toggle between insert and replace mode.
           First the line editor is in insert mode.
[CTRL-A] = Dependent on the called command.
[CTRL-L] = Move right one character
[CTRL-E] = Move to end of line
[CTRL-H] = Move one character left
[CTRL-B] = Move to begin of line
[CTRL-D] = Delete character under cursor
[RUBOUT] = Delete one character to the left
[CTRL-\] = Delete character under cursor to end of line
[CTRL-O] = Delete whole line
```

The cursor need not be at the end of the line when the [CR] is entered.

### 1.1.6 Program or Command Abort

There are two basic methods of aborting a running program or command.

The first one is the ABORT switch on the CPU-board. This switch causes a level 7 interrupt to the processor. If a VMEPROM command was under execution at this time, the message "Abort switch pressed" is displayed and control is transferred back to the command interpreter immediately.

If a user program is running when the ABORT switch is pressed, the current contents of the processor registers are saved and a message along with the processor registers is displayed.

The second method is typing ^C twice on the keyboard. If that happens, VMEPROM will abort the current command or program within 1.28 seconds and control is transferred to the command interpreter. The processor register is not saved by this action. They show the same status as they had before the program was started.

### 1.1.7 Command or Batch Files

If command or batch files are executed, the parameters from the command line can be used. The ’&’ character is used for character substitutions. ’&0’ is replaced with the last system error number. ’&1’ is replaced with the first parameter of the command line, ’&2’ with the second, and so forth up to ’&9’. ’&#’ is replaced with the current task number.

```
Example: ? SF DOIT
         RM &1 &3
         RM &2 &4

         ? DOIT D0,A1,12345678,1000
         ? RM D0 12345678
         ? RM A1 1000
```

## 1.2 VMEPROM Built-in Commands

The VMEPROM built-in commands are described in detail in this chapter.

The following notations are used throughout this document:

- Symbolic representation is put in arrows (i.e. <address> where an absolute address has to be inserted, or <filename> where a filename has to be inserted.

- Optional arguments are in square brackets (i.e. [<option>]). Those arguments must not be specified and have a default value.

- If one argument out of more can be selected, the arguments are separated by a "|" (i.e. [B | W | L] to select Byte, Word or Long Word size).

- If more than one out of many possibilities for an argument has to be selected, these are marked with a "&" sign (i.e. [B|W|L&N&O|E] to select B or W or L together with N and O or E).

Some more hardware related commands may be available. These commands are described in detail in the User’s Manual of your particular CPU board.

Most of the VMEPROM commands assume that the parameters are given in hex (without a leading $ sign).

However, some values are assumed in decimal.

These are:

**Port** — VMEPROM port numbers are in the range 0-15 and have to be entered in decimal.

**Disk** — The disk numbers have to be entered in decimal

**Level** — The directory levels have to be entered in decimal

**Tasks** — The task numbers have to be entered in decimal

**Task Priorities** — The task priority has to be entered in decimal

**Error Numbers** — The error numbers are displayed in decimal and have to be entered in decimal

### 1.2.1 # - Symbolic Command Name

```
Format: #
        # <name>
        # <name>,<command string>
```

The symbolic name command is used to display, delete or define a symbolic name for often used command lines. The first format displays all currently defined names, the second deletes a defined name from the list and the third one defines a new name with the command string. VMEPROM supports up to 5 symbolic names with command lines of up to 40 characters.

Symbolic names can reference other symbolic names.

Example:

```
? # ASM AS 8000  Define ASM for the command AS

? # DISP DN      Define DISP for display disk name

? # D DISP       Define D for DISP

? #              Show defined symbolic names
ASM: AS 8000
DISP: DN
D: DISP

? DISP
Disk 8: SY$STRT

? D
Disk 8: SY$STRT

? ASM
8000             : XEXT
                 : _
```

### 1.2.2 <u>AF - APPEND FILE</u>

Format: AF <file1>,<file2>

The APPEND FILE command concatenates two files. The first file <file1> is appended onto the end of file <file2>. The file type attribute of <file1> is transferred to <file2>. The contents of <file1> is not affected by the operation.

A [CTRL-C] interrupts this function on a sector boundary, closes both files, and returns to the monitor. This action is reported by the message '^C'.

The APPEND FILE command uses the assembly primitive XAPF.

Example:

```
? AF temp1,temp2   Append file temp1 to the end of temp2
? _
```

### 1.2.3 AS - LINE ASSEMBLER

Format: AS <address>

The AS command invokes the line assembler/disassembler of VMEPROM. It can assemble and disassemble all 68000/010 instructions and all the PDOS system calls listed in section 4 of this manual. In addition the 68020/68030 version of VMEPROM can assemble and disassemble all 68020/68030 and 68881/68882 opcodes.

The AS command, when invoked, displays the current address offset and the address within the window. Then the current location is disassembled.

After the prompt on the next line, the user can enter one of the following:

1) A valid 680x0 mnemonic. Some 68020/68030 addressing modes allow omission of arguments. These addressing modes can be entered by omitting the argument and typing the dividing character ','.

```
Examples: CLR.W ([$1,A0],D0.W,$2)
          CLR.W ([$1,A0],,$2)
               CLR.W ([,A0],,)
```

2) A '#' sign followed by the new address within the window. This is an absolute address change.

3) An '=' to disassemble the same location again.

4) A '+' to disassemble the next location.

5) A '-' sign forces the disassembler to step back one possible opcode. If none is found the same location will be opened.

6) A '+' or '-' sign followed by the number of bytes the address has to be increased or decreased. This is a relative address change.

7) A '.' or [ESC] to exit the line assembler and return control to the command interpreter.

8) [CTRL-A] to edit the disassembled opcode.

All immediate values, addresses and offsets inside mnemonics are assumed to be entered in decimal. So hex values have to be proceeded with a dollar ($) sign. In addition, binary values may be used if proceeded by a percent sign ("%") and octal values if proceeded by an at/around sign ("@"). The disassemblers display all values in hex representation.

The line assembler accepts a pseudo opcode of the form DC.B, DC.W and DC.L to define constant data storage. An ASCII pattern can be stored by using DC.B with the format DC.B "ASCII. All characters after '"' will be written to memory. The disassembler displays all illegal opcodes as DC.W.

Both the line assembler and disassembler support the opcodes as described in Chapter 4 of the VMEPROM Manual.

Example:

```
? AS 8000
8000      : XEXT
          : MOVE.L #$123,D1  New opcode entered
8006      : ORI.B #0,D0
          : -                Step back one opcode
8000      : MOVE.L #$123,D1
          : [CTRL-A]         Recall line
          : MOVE.L #$1234,D1 Line edited
8006      : ORI.B #0,D0
          : XRDM             New opcode
8008      : ORI.B #0,D0
          : -8               Back 8 bytes
8000      : MOVE.L #$1234,D1
          : +                Disassemble next instruction
8006      : XRDM
          : [CR]             Disassemble next instruction
8008      : ORI.B #0,D0
          : #8010            Go to absolute address $8010
8010      : ORI.B #0,D0
          : .                Back to the command
                             interpreter

? _
```

### 1.2.4 **ASSIGN - Assign New Input or Output Ports**

```
Format: ASSIGN <port>
        ASSIGN <port>,<output port>
```

The ASSIGN command has two functions, depending on the command line arguments. If the output port is omitted, ASSIGN sets a new input and output port for the current task. If the output port is specified, the default input/output ports are unchanged, but the alternate output ports of the task are changed. The output port specified must be in the range 1-4.

Example:

```
? ASSIGN 3                VMEPROM now uses port 3 for I/O

? ASSIGN 3,2              Use port 3 as unit 2 port
```

### 1.2.5 BASE - SET/DISPLAY BASE REGISTER

Format: BASE
BASE <address>

The BASE register in VMEPROM is used to offset all memory accesses into the tasks memory. So all debugging can be done relative to address 0, which is actually the begin address of your tasks memory. This saves a lot of typing and makes sure that no other tasks memory is destroyed by a typing error.

Example:

```
? base<cr>                            Display BASE register
Base = 00000000  : <cr>               No changes

? base 8000<cr>                       Set BASE register to $8000
? base<cr>                            Display BASE register
Base = 00008000  : <cr>

? M 0<cr>                             Open   address   $0   +BASE
register
8000+0000   A00E : <cr>
8000+0002   0000 : <cr>
8000+0004   0000 : .

?
```

### 1.2.6 BENCH - Built-in Benchmarks

```
Format: BENCH
        BENCH <#>,<address>
```

These function can execute one of the built-in benchmarks. If only BENCH is entered, a short descriptions of all benchmarks is displayed on the terminal. A benchmark is executed by entering the number of the benchmark (in decimal) and the address where it shall run in memory (in hex).

The following benchmarks are available:

```
Bench  1:  Decrement long word in memory, 10.000.000 times
Bench  2:  Pseudo DMA 1K bytes, 50.000 times
Bench  3:  Substring character search, 100.000 times, taken from EDN,
           08/08/85
Bench  4:  Bit Test/Set/Reset, 100.000 times, taken from EDN,08/08/85
Bench  5:  Bit Matrix Transposition, 100.000 times, taken from EDN,
           08/08/85
Bench  6:  Cache test, executes 128K bytes program 1000 times
           CAUTION: This benchmark will destroy 128K bytes memory
Bench  7:  Floating Point - 1.000.000 Additions
Bench  8:  Floating Point - 1.000.000 Sines
Bench  9:  Floating Point - 1.000.000 Multiplications
Bench 10:  100.000 Context switches
Bench 11:  100.000 Set system event
Bench 12:  100.000 Change task priority
Bench 13:  100.000 Send and Receive task message
Bench 14:  100.000 Read system time
```

Example:

```
? bench 1 8000     Execute benchmark #1 at address $8000

Bench  1: Decrement long word in memory, 10.000.000 times
Benchmark time = 0:07.23

?
```

### 1.2.7 BF - Block Fill

```
Format: BF <begin>,<end>,<value>,[B | W | L]
        BF <begin>,<end>,<pattern>,P
        BF <begin>,<end>,<opcode>,O
```

This command fills the specified memory area with a constant. The type of the constant is defined by the option and may be a Byte, Word, Long word, Pattern or an Opcode. A pattern is an ASCII string which is to be put in inverted commas. The maximum length is only restricted by the length of the input line, which may not exceed 78 characters. An Opcode is each valid 680x0 mnemonic or an opcode as described in Chapter 4 of the VMEPROM Manual. If the pattern or the opcode contains argument seperators, such as space, comma, or full stop, the data has to be put in brackets. If no option is specified, a default of Word is assumed.

Example:

```
? BF 8000 8100 NOP O

? MD 8000 10
00008000: 4E 71 4E 71 4E 71 4E 71  4E 71 4E 71 4E 71 4E 71 NqNqNqNqNqNqNqNq

? BF 8000 8100 ("Hello World") P

? MD 8000 10
00008000: 48 65 6C 6C 6F 20 57 6F  72 6C 64 48 65 6C 6C 6F Hello World Hello

? BF 8000 8100 12345678 L

? MD 8000 10
00008000: 12 34 56 78 12 34 56 78  12 34 56 78 12 34 56 78 .4Vx.4Vx.4Vx.4Vx

? BF 8000 8100 &255

? MD 8000 10
00008000: 00 FF 00 FF 00 FF 00 FF  00 FF 00 FF 00 FF 00 FF ................

? _
```

### 1.2.8 BM - Block Move

Format: BM <begin>,<end>,<destination>

The BM command copies a memory from one area to another. The areas may be overlapped.

Example:

```
? MD 8000 20
00008000: 00 FF 00 FF 00 FF 00 FF  00 FF 00 FF 00 FF 00 FF ................
00008010: 00 FF 00 FF 00 FF 00 FF  00 FF 00 FF 00 FF 00 FF ................

? BM 8000 8020 9000

? MD 9000 20
00009000: 00 FF 00 FF 00 FF 00 FF  00 FF 00 FF 00 FF 00 FF ................
00009010: 00 FF 00 FF 00 FF 00 FF  00 FF 00 FF 00 FF 00 FF ................

? _
```

### 1.2.9 BOOT - Booting an Operating System

```
Format:  BOOT
         BOOT <P|U|O|V>
```

The BOOT command allows the user to boot an operating system (PDOS/UNIX/Another operating system) or to do the initialization for working with the VMEPROM-UNIX communication.

After the initialization of VUCP, a task will be connected to UNIX-CPU via a RAM port.

To boot an operating system a bootstrap must be available. Otherwise an error message will occur.

Example:

```
? BOOT P

  Force PDOS Bootstrap, Revision x.y (date)
        Disk #0 : FORCE PDOS was found>> Sector 2336, Addr $800
                  Execute it? Yes. Booting....SUCCESS!
```

### 1.2.10 <u>BP - BAUD PORT</u>

```
Format: BP
        BP <port #>
        BP {-}<port #>,<baud rate>
        BP {-}<port #>,<baud rate>,<type>,<UART base addr>
```

The BAUD PORT command initializes a VMEPROM I/O port and binds a physical UART to a character buffer. The command sets the UART character format, receiver and transmitter baud rates, and enables receiver interrupts.

The first parameter <port #> selects the console port in ranges from 1 to 15. This corresponds to character input buffers defined in the VMEPROM system RAM (SYRAM). If a minus (-) precedes the port number, then the associated port # is stored in the UNIT 2 (U2P$(A6)) variable.

Receiver and transmitter baud rates are initialized to the same value according to the <baud rate> parameter. The <baud rate> parameter ranges from 0 to 8 or the corresponding baud rates of 19200, 9600, 4800, 2400, 1200, 600, 300, 110, or 38400. Either parameter type is acceptable.

Baud Rates Allowed:

```
0 = 19200 baud
1 = 9600  baud
2 = 4800  baud
3 = 2400  baud
4 = 1200  baud
5 = 600   baud
6 = 300   baud
7 = 110   baud
8 = 38400 baud
```

The <type> and <UART base addr> are optionally included when binding a logical port to a different UART. For <type> information, refer to the User’s Manual of your CPU-board.

The <port #> can also be used to set or reset the port flags. These are bit positions 8 through 15 of the resulting integer value and are defined to the right. It is recommended that hex format be used when setting these parameters.

```
$100 + port  = CtrlS CtrlQ protocol
$200 + port  = Pass control characters
$400 + port  = DTR protocol
$800 + port  = 8-bit character I/O
$1000 + port = receiver interrupts disable
$2000 + port = even parity enable
$4000 + port = clear flag bits
```

If the BP command has no arguments, a listing of all currently installed ports is sent to the console. ’Task’ parameter indicates the currently assigned task to that port.

Example:

```
? BP
Port  Type   fwpi8dcs    Base    Baud    task
# 1     1    00001100  FF800000  9600      1

? BP 2,1,1,$FF800200   Initialize the UART
?
```

### 1.2.11 BR - Set/Display/Delete Breakpoints

```
Format: BR
        BR *
        BR <number>
        BR <number>,<address>
        BR <number>,<address>,<command>
        BR <number>,<address>,[<command>],<count>
```

VMEPROM supports a maximum of 10 breakpoints in the range 0-9. The BR command is used to set, display or delete breakpoints.

The first format displays all currently defined breakpoints. The second one deletes all defined breakpoints. The third format is used to delete one single breakpoint. The other formats are used to define one breakpoint. If a breakpoint is already defined it will be overwritten. Two breakpoints looking for the same address are not possible.

If a count is specified, the program first stops at the breakpoint when this specification has been achieved. The default value is one.

The default action taken by a breakpoint is a display of the breakpoint number encountered and a display of all processor registers.

So there is a fourth option of the command line to change the default behaviour at a breakpoint. The command, which can be specified is executed instead of the display described before. The command may not have any arguments and may have a length of up to 9 characters.

The command may be a symbolic name, one of the built-in commands of VMEPROM, an installed utility or a disk file (command file or program).

Example:

```
? BR 0 8020             Define breakpoint 0 at address $8020

? BR
Defined Breakpoints:
 B0  $8020  1

?
```

### 1.2.12 <u>BS - Block Search</u>

```
Format:  BS <begin>,<end>,[/]<value>[,<option>]
         BS <begin>,<end>,[/]<pattern>,P
         BS <begin>,<end>,[/]<opcode>,O
```

This command searches the specified memory area for a constant. The type of the constant is defined by the option and may be a Byte, Word, Long word, Pattern, or an Opcode. A pattern is an ASCII string which is to be put in inverted commas. The maximum length is only restricted by the length of the input line, which may not exceed 78 characters. An Opcode is each valid 680x0 mnemonic or an opcode as described in Chapter 4 of the VMEPROM Manual. If the pattern or the opcode contains argument separators, such as space, comma, or full stop, the data has to be put in brackets. If no option is specified, a default of Word is assumed.

The data which has to be searched in memory may be preceded by a '/' to look only for locations not containing the value, pattern or opcode.

Example:

```
 ? BS 8000 8100 /1234           Search memory for "not" value
 Search:  8020    = 5678          Found

 ? BS 8000 8100 5678              Search memory for value
 Search:  8020    = 5678          Found

 ? BS 8000 8100 ("Hello World") P Search memory for pattern
                                  None found
 ? BS 8000 8100 (ADDQ.L #1,D0) O  Search memory for opcode
                                  None found
 ? _
```

### 1.2.13 <u>BT - Block Test</u>

Format: BT <begin>,<end>

The Block Test command performs an in-depth memory test within the specified address limits. The following passes are performed:

1) Byte Pattern Test
2) Word Pattern Test
3) Long Pattern Test
4) Word Shift Test
5) Address Test

If any errors are found they are reported with the type of test which failed, the address and the differing values. In addition the error counter in the task control block (TCB) is incremented.

Example:

```
? bt 200000 300000      Test memory from $200000 to $300000
?
```

### 1.2.14 **BV - Block Verify**

Format: BV <begin>,<end>,<destination>

This command compares two blocks of memory. If the specified blocks are not equal, the different values and the memory location is displayed. In addition the error counter in the task control block (TCB) is incremented.

Example:

```
? bv 8000 8080 8080
Verify:  8021    = 70  80A1    = 71

?
```

### 1.2.15 <u>CF - COPY FILE</u>

```
Format:  CF <file1>,<file2>[[,<file3>,<file4>],...]
         CF <file1>,<disk#>[[,<file2>,<file3>],...]
```

The COPY FILE command copies file1 into file2. If the second parameter is a disk number the file is copied to this disk. If the destination file exists it is destroyed and replaced by the source file. Otherwise the destination file will be created. The file type attribute is transferred. The source file is not affected by the operation.

A [CTRL-C] interrupts this function on a sector boundary, closes both files, and returns to VMEPROM. This action is reported by the message '^C'.

Example:

```
? lc
  test          lv            ls            lc
Number of files: 4            Sectors allocated:  5

? cf test,test1
? lc
  test          lv            ls            lc            test1

Number of files: 5            Sectors allocated:  6

?
```

### 1.2.16 COLD - Cold Start VMEPROM

Format: COLD

The COLD command is used to reinitialize all VMEPROM variables. It takes the same action as a reset, except that the kernel and all associated tasks are not affected.

Example:

```
? COLD
*******************************************************************
*                                                                 *
*                         V M E P R O M                           *
*                                                                 *
*          SYS68K/CPU-xx     Version  a.bb      Date              *
*                                                                 *
*           (c) FORCE Computers  and  Eyring Research             *
*                                                                 *
*******************************************************************

? _
```

### 1.2.17 **CONFIG - READ HARDWARE CONFIGURATION**

Format: CONFIG

The CONFIG command searches for the available hardware configuration on the VMEbus. This function is implementation dependant.

For details please refer to the User’s Manual of you CPU-board.

If you are using Winchester disks, please make sure that the disk drive is up to speed when the CONFIG command is executed.

The CONFIG command also installs the loadable driver for all boards which are available.

Example:

```
? CONFIG
Disk driver FORCE-ISCSI1 installed
UART FORCE-ISIO1 installed
ISCSI-1 :  1 boards available
 ISIO-1 :  1 boards available

?
```

### 1.2.18 CREATE TASK

```
Format:  CT <command>,<size>,<[time*256+]priority>,<port>
         CT ,<size>,<[time*256+]priority>,<port>
         CT <address>,<size>,<[time*256+]priority>,<port>
```

The CREATE TASK command places a new task entry in the task queue and lists the realtime kernel of VMEPROM. Parameters for the new task include a command line, memory size, task priority/time slice, and an I/O port. The new task number is reported after task creation.

The <command> parameter is the command line for the new task. The string is passed to the new task via a message buffer and cannot exceed 64 characters in length.

Multiple commands and parameters are passed by using parentheses.

If the first parameter is omitted, then the VMEPROM monitor is invoked.

If an address is specified instead of <command>, this address is interpreted as the start address of a program in memory. The address must be specified in hexadecimal and start with a number 0-9 not to conflict with a program name.

The amount of memory for the new task is given by <size> and is in 1 Kbyte increments (although rounded to the next 2 Kbyte boundary). The minimum amount of memory is 8 Kbyte. The system memory bit map is searched for a contiguous block of memory equal to <size>. If the search fails to find a large enough block, then memory is taken from the parent task and allocated to the new task.

The <priority> parameter specifies the new tasks priority. The range of task priority is from 1 to 255 where 255 is the highest priority. The highest priority, ready task always executes. Tasks on the same priority level are scheduled in a round robin fashion. The time a task is in running state is also given with the <priority> parameter. If no time is specified it will default to one time slice. Otherwise it is calculated to "time*256+priority".

The <port> parameter assigns an I/O port to the new task. Port 0 is the default and is called the phantom port. On the phantom port, all character outputs and conditional inputs are ignored while requests for character input result in the task aborting with error 86. More than one task may be assigned to an output port. The input port is a cannot be shared with another task. Input ports are allocated on a first come basis. No VMEPROM monitor task with the phantom port (port 0) can be created.

After a task is created, the spawned task number is reported. This number is used in killing the new task.

The values for size, priority and port have to be entered in decimal.

Example:

```
? LT
task   pri   tm  ev1/ev2  size     pc       tcb      eom      ports    name
*0/0    64    1            700  FF01FAB8 00007000 000B6000  1/1/0/0/0  LT

? CT ,100,64,2
 Sontask number = 1

? CT TEST,20,,0
 Sontask number = 2

? CT 1000,,2560,0
 Sontask number = 3

? LT
task   pri   tm  ev1/ev2  size     pc       tcb      eom      ports    name
*0/0    64    1            572  FF01FAB8 00007000 00096000  1/1/0/0/0  LT
 1/0    64    1       98   100  FF002986 0009D000 000B6000  2/2/0/0/0
 2/0    63    1             12  00099000 00098000 0009B000  0/0/0/0/0  TEST
 3/0    64   10              8  00010000 00096000 00098000  0/0/0/0/0

? _
```

### 1.2.19 DD - Disk Dump

Format: DD <disk>,<sector>
DD <disk>,<sector>,<count>

The disk dump command displays the raw contents of disk sectors on the terminal. An optional count specifies the number of contiguous sectors to be dumped.

The data is represented in hex and ASCII.

The DD command expects the disk number and the count to be entered in decimal while the sector number is assumed to be in hex.

Example:

```
? dd 0 0 1

Disk # 0   Sector = 0 ($0)

0000  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0010  00 00 00 0E 00 00 00 00  00 80 09 20 A5 5A FF FF  ........... .Z..
0020  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
0030  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
0040  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
0050  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
0060  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
0070  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
0080  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
0090  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
00A0  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
00B0  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
00C0  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
00D0  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
00E0  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
00F0  FF FF FF FF FF FF FF FF  FF FF FF FF FF F8 00 00  ................
More (cr) ? <esc>

?
```

### 1.2.20 <u>DF - DEFINE FILE</u>

```
Format: DF <file{;level}{/disk}>
        DF <file{;level}{/disk}>,<sectors>
```

The DEFINE FILE command creates a new file in a disk directory. <File> specifies the file name, and if included, {;level} the file directory level and {/disk} the disk directory number. Defaults for the latter two parameters are the current level and disk number.

The <sectors> parameter specifies the number of contiguous sectors to allocate to the file. One initial sector is allocated if the <sectors> parameter is not specified. Only contiguous files can be defined. A contiguous file facilitates random access to the file data since VMEPROM can directly position to any byte within the file without following sector links. The <sectors> parameter is expected to be given in decimal.

If a contiguous file is extended past the original allocation length and a non-contiguous sector is appended to the file, then the contiguous file attribute is deleted.

Therefore, even though contiguous files can be extended, you should allocate enough sectors when the file is first defined to handle all anticipated data. Otherwise, random file access slows down.

The length of a contiguous file is specified in sectors. Each sector contains 252 bytes or characters of data. The file size is given by the number of sectors times 252. The maximum file size is limited by the size of the logical disk.

Example:

```
? DF df1
? LC

  df1
Number of files: 1            Sectors allocated:  1

?
```

### 1.2.21 DI - Disassembler

```
Format: DI <address>
        DI <address>,<count>
```

The DI command causes the disassembler to be invoked and display the mnemonic, starting at the specified address. If count is specified, it is interpreted as the number of lines to display. If count is omitted, a full page is displayed on the terminal and the user is then prompted to continue disassembly (enter <cr>) or to return to the command interpreter (enter any other key).

The disassembler supports all 68000/010 mnemonics. The 68020/68030 version of VMEPROM also supports the 68020/68030 and the 68881/68882 opcodes.

Example:

```
? DI 8000 5

8000   NOP
8002   NOP
8004   NOP
8006   NOP
8008   NOP

?
```

### 1.2.22 DL - DELETE FILE

```
Format:  DL <file>
         DL <file1>,<file2>,...
```

The DELETE FILE command removes from the disk directory the specified file(s). All sectors associated with that file(s) are deallocated in the disk's sector bit map and freed for use by other files on the same disk. A file cannot be deleted if it has previously been either delete- or write-protected.

These protection flags must be removed with the 'SA' command before the file can be deleted from the disk.

A sector bit map is maintained by VMEPROM on each disk so that file creation and deletion does not require a disk compaction routine to recover lost disk space.

However, frequent file definitions, deletions, and extensions do create small groups of contiguous sectors which tend to fracture files and make the creation of contiguous files impossible. This is remedied by periodically transferring all files to a newly initialized disk which allocates sectors sequentially for each file.

Example:

```
? lc
  df1           df2           temp          df3           dl1
Number of files: 5            Sectors allocated:  14

? dl temp
? lc
  df1           df2           df3           dl1
Number of files: 4            Sectors allocated:  5

?
```

### 1.2.23 DN - Display/Change the name of a disk

```
Format: DN
        DN <disk#>
        DN <disk#>,<name>
```

The DN command displays or changes the name of a logical disk. If the disk number is omitted, the current system disk is assumed. If no name is given, the current name is displayed, if a name is specified it is assigned to the disk. The disk name is only for "human" readers and is not used by any of the VMEPROM commands.

Example:

```
? DN 6
Disk 6: VMEPROM DOC

? DN 0 Test disk
? DN 0
Disk 0: Test disk

?
```

### 1.2.24 DR - Display Processor Registers

Format: DR [T] [-M]

The DR command displays processor registers. The displayed registers are not real current processor registers, but those kept in memory and loaded to the processor when a program is started. When program execution is terminated (XEXT instruction, trap or breakpoint or other exception) the processor registers are resaved and can be displayed by the DR command.

When choosing the option 'T', only the program counter, stack pointer, and address registers A5 and A6 will be displayed until 'T' is used a second time. Then all registers will once again be displayed. First VMEPROM is configured to display all registers.

The option '-M' is used to additionally display the MMU registers if an MMU is available.

See also: 1.2.25 DRF - Display floating point registers

**Note:** Registers chosen for displayal are processor dependent.

Example:

```
? DR
       0        1        2        3         4        5        6        7
D: 00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
A: 00000000 00000000 00000000 00000000  00000000 00001000 00007000 0009AFFC

 VBR = 00000000   CAAR = 00000000   CACR = 00000001   SFC  = 0   DFC = 0
*USP = 0009AFFC   SSP  = 00007BE6   MSP  = 000078C4
 PC  = 00008000   SR   = 0000 ..U..0........

? DR T
PC = 00008000  SP = 0009AFFC  A6 = 00007000  A5 = 00001000

? DR
PC = 00008000  SP = 0009AFFC  A6 = 00007000  A5 = 00001000

? DR -M
PC = 00008000  SP = 0009AFFC  A6 = 00007000  A5 = 00001000

MMU_SR  = EE47     -> BLS.WIM..T...007
MMU_TC  = 03FFFFFF
MMU_TT0 = FFFF0777            MMU_TT1 = FC038514
MMU_CRP = B46B7228_80020000  MMU_SRP = 252D3368_00001003

? DR T
       0        1        2        3         4        5        6        7
D: 00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
A: 00000000 00000000 00000000 00000000  00000000 00001000 00007000 0009AFFC

 VBR = 00000000   CAAR = 00000000   CACR = 00000001   SFC  = 0   DFC = 0
*USP = 0009AFFC   SSP  = 00007BE6   MSP  = 000078C4
 PC  = 00008000   SR   = 0000 ..U..0........
? _
```

### 1.2.25 DRF - DISPLAY REGISTERS OF THE 68881/68882

Format: DRF

This command displays the registers of the 68881/68882 coprocessor. Like the processor registers, these registers are saved and restored whenever a user program is invoked. This command gives an error if no 68881/68882 coprocessor is installed.

See also: 1.2.24 DR - DISPLAY REGISTERS

**Note:** This command is only available for 32 bit processors.

Example:

```
? DRF

FP0: 0.00000000 E+000 0.00000000 E+000 0.00000000 E+000 0.00000000 E+000
FP4: 0.00000000 E+000 0.00000000 E+000 0.00000000 E+000 0.00000000 E+000
```

### 1.2.26 **DT - DATE AND TIME**

Format: DT

The DT command outputs the current date and time to the user console. These values can be changed by the ID command.

Example:

```
? DT
16-Mar-88
16:47:38

?
```

### 1.2.27 DU - Dump S-record

```
Format: DU <begin>,<end>
        DU <begin>,<end>,<command line>
```

This command sends an S-Record to the standard output port. It may be redirected with the usual redirection method.

An optional command line may be specified which is sent via the output port before the S-record starts. This can be used to start a load command on the host system.

The following S-record types are supported:

| | |
|---|---|
| S1 | Start record |
| S2 | Data record, this type is needed if the end address is smaller than $8000. |
| S3 | Data record, this type is used if the end address is bigger than $800000. |
| S7 | End-record for S3 records. |
| S8 | End-record for S2 records. |
| S9 | End-record for S1 records. |

The address field of all End-records is 0.

Example:

```
? DU 8000 8020
S0030000FC
S2180080004E714E714E714E714E714E714E714E714E714E71F1
S2100080144E714E714E714E714E714E71E1
S804000000FB

? DU 8000 8020 >2
?
```

### 1.2.28 <u>ED - VMEPROM Screen Editor</u>

Format: ED
        ED <filename>

The ED command invokes the build in screen editor of the VMEPROM.

An existing file can be specified at the command line and will be loaded when the editor starts.

The size of the editing file depends on the size of the tasking memory where the editor works. The editor always works in the character insert mode with a maximum line size of 79 characters. When the line size is exceeded the cursor automatically wraps to the next line. If there is still space in the edit buffer, a new line will be inserted. The screen holds up to 22 (0-21) text lines. Line 22 is left blank and line 23 is the status line. The status line holds the current cursor position and is used for displaying messages and receiving inputs for some commands.

Note : The ED only can work correctly if the terminal is installed with the 'ST' command.

Editor Commands:

1. Help and Status

| | |
|---|---|
| <CTRL>A | Display the on-line help screen. |
| <ESC>A | Display editor status information. |

2. Cursor Movement

| | |
|---|---|
| <CTRL>H | Moves the cursor one character position left but does not wrap to the previous line when the left screen side is reached. |
| <CTRL>L | Moves the cursor one character position right but does not wrap to the next line when the right screen side is reached. |
| <CTRL>J | Moves the cursor one line down. |
| <CTRL>K | Moves the cursor one line up. |
| <CTRL>B | Moves the cursor to the beginning of the current line. |
| <CTRL>E | Moves the cursor to the end of the current line. |
| <CTRL>U | Moves the cursor one page upward and centers the screen. |
| <CTRL>N | Moves the cursor one page down and centers the screen. |
| <CTRL>T | Moves the cursor to top of file. |
| <CTRL>Z | Moves the cursor to end of file and centers the screen. |

3. Text editing

| | |
|---|---|
| <DEL> | Deletes one character left from the current cursor position and wraps to the previous line when reaches the left screen boundary. |
| <CTRL>D | Deletes one character at the current cursor position and merges the following line to the current when it is pressed at the end of the current line. |
| <CTRL>O | Deletes the current line. |
| <CTRL>\ | Deletes from the cursor position to the end of the current line including the character at the cursor position. |

4. Line Buffer

| | |
|---|---|
| <ESC>G | Get the current line into the line buffer without changing the current line. |
| <ESC>S | Swap the line in the line buffer against the current line. |
| <ESC>I | Insert the line in the line buffer before the current line. |

5. Text Pattern search

| | |
|---|---|
| <CTRL>F | Find a text pattern, center screen and place cursor at the end of the found pattern. |
| <CTRL>P | Repeat last pattern search. |

6. File Operations

| | |
|---|---|
| <CTRL>G | Get a file from the disk and reinitialize the editor. |
| <CTRL>W | Write the edit buffer contents to a disk file. An existing file will be overwritten. |

7. Other Functions

| | |
|---|---|
| <CTRL>I | Insert TAB at current cursor position. |
| <CTRL>] | Set TAB spacing (default is every 8th column). |
| <CTRL>R | Character repeat function. Allowed keys are any printable character and <DEL>. |
| <CTRL>V | Restarts the editor. All existing text and initializations are lost. |
| <ESC>Q | Quits the editor and returns to VMEPROM. |

### 1.2.29 ER - LIST ERRORS

```
Format: ER [-c]
        ER 0 [-c]
        ER <error#>
```

The LIST ERROR command has three functions. The first one, with no argument, displays the number of errors found on one of the following commands:

1) Block Test
2) Block Verify
3) Block Search.

The second format, with the argument "0" resets the above error count to 0.

If the optional parameter [-c] is given when using the first two formats, an execution count will be displayed or reset to zero. The execution count will be incremented before it is displayed.

The third format requires a valid error number as an argument and displays the VMEPROM error message associated with <error#>.

Error numbers range as follows:

| | |
|---|---|
| VMEPROM errors | 1- 49 |
| PDOS errors | 50- 99 |
| Disk errors | 100-299 |

Example:

```
? ER
Current error count = 6

? er 0

? er 2
Command line argument error

?er 0 -c

?er -c
 Current error count = 0    Execution count = 1
```

### 1.2.30 EV - SET/RESET EVENT

```
Format: EV
        EV {-|+}<event>
        EV {-|+},<address>,<bit#>
```

VMEPROM events are set, reset, or listed with the EV command. Both logical and physical events can be accessed with EV. The delayed event queue can also be listed or cleared with the EV command.

If the first parameter is zero, the delay queue is cleared. For accessing a logical event, the event number <event> has to be entered. If <event> is proceeded by a plus (+) sign, the event is set and the old status is returned. If <event> is proceeded by a minus (-) sign, the specified event is cleared and its old status is displayed. For accessing a physical event, the second parameter must be the byte address followed by the bit number (0-7), where bit 7 is the most significant bit of the byte. Physical events are set (+), reset(-) and list(_) in the same way as logical events are accessed. If no special sign is specified, the current status of the event is displayed. If <event> is omitted, a status list of all events in the system and all pending delay events are displayed.

The event number has to be entered in decimal.

Current logical event definitions are as follows:

```
  1-63 = Software events                120 = Level 2 lock
 64-80 = Software resetting events      121 = Level 3 lock
 81-95 = Output port events             122 = Batch event
96-111 = Input port events              123 = Spooler event
   112 = 1/5 second event               124 = Reserved
   113 = 1 second event                 125 = Reserved
   114 = 10 second event                126 = Reserved
   115 = 20 second event                127 = Virtual ports
   116 = Reserved                       128 = Local event
   117 = Reserved
   118 = Reserved
   119 = Reserved
```

Example:

```
? EV
  00000000  00000000  00000000  0000FE00
  EV 128 : TASK 0  SET DELAY = 43 TICS

? EV 10
  Is  0

? EV +10
  Was 0

? EV -10
  Was 1

? EV 10
  Is  0

? EV +,$10000,1
  Was 0

? EV, $10000,1
  Is 1
```

### 1.2.31 FD - File Dump

Format: FD <file>

The File Dump command dumps the contents of a file on the terminal.

The file contents is displayed in hex and ASCII representation.

Example:

```
? FD test
0000  54 68 69 73 20 69 73 20  61 20 73 61 6D 70 6C 65   This is a sample
0010  20 66 69 6C 65 2E 20 49  74 20 77 61 73 20 63 72    file. It was
cr
0020  65 61 74 65 64 20 75 73  69 6E 67 20 74 68 65 20   eated using the
0030  4D 46 20 63 6F 6D 6D 61  6E 64 20 6F 66 20 74 68   MF command of
th
0040  65 20 56 4D 45 50 52 4F  4D 0D                     e VMEPROM.

? _
```

### 1.2.32 FM - FREE MEMORY

```
Format:  FM
         FM -E
         FM {-}<size>
```

The FREE MEMORY command drops memory from your current task.

If no parameter is given all free memory contiguous to tasking memory is displayed.

If parameter '-E' is given all free memory is displayed. This includes memory which is not contiguous to tasking memory but deallocated in the memory bit map.

If the <size> parameter is positive, then the memory is deallocated and made available to the system for other task usage. If the <size> parameter is negative, then the memory is simply dropped from the current task and is not recoverable. The size parameter must be entered in decimal.

Example:

```
? FM
No free memory contiguous to tasking memory

? FM -E
Free memory:    2 kbyte at $B6000

? FM 100
100 Kbytes free at address $9C800

? FM
Free memory: 100 Kbyte

? FM -10
10 Kbytes free at address $9A000

? FM
No free memory contiguous to tasking memory

? FM -E
Free memory:  100 kbyte at $9C800
Free memory:    2 kbyte at $B6000

? _
```

### 1.2.33 FRMT - Format Floppy or Winchester Disk

Format: FRMT

FRMT - DISK HARDWARE FORMAT

Caution: FRMT may only be run when no other tasks are running. The hardware configuration must be checked before this command can be executed (See CONFIG command).

Description: FRMT allows you to define drives and to format and partition disk drives. VMEPROM supports one floppy and up to three Winchester drives for a maximum of four disk controllers.

When you run this command, you may select a drive to access (i.e. F, F0-F8 for the floppy diskette drives or W, W0-W15, for up to 16 Winchester drives). Enter the drive letters followed by a [CR] to access the drive. Please note that all entries must be in upper case letters. If the drive is undefined, you will be prompted with the drive select byte for the controller.

```
?FRMT

68K VMEPROM FORCE Format Drive Utility 16/03/88

Possible Disk Controllers in this System are:

          Controller #1 is not defined
          Controller #2 is a FORCE WFC-1
          Controller #3 is a FORCE ISCSI-1

Drives that are currently defined in system are:

F0 is controller #3, drive select byte $73
F1 is controller #3, drive select byte $74
W0 is controller #3, drive select byte $00
All not named drivers are undefined.

Select Menu:  W,W0-W15=Winch; F,F0-F8=Floppy; Q=Quit
Select Drive:  _
```

If you select either a floppy drive or a Winchester drive that is already defined, FRMT directly enters the Drive Command Menu. If you are installing a new Winchester drive which is currently undefined, then you must enter the controller number and drive select jumpering (0-3).

The Drive Command Menu tells you which drive you are currently dealing with and has the following commands:

```
 Select Menu :  W,W0-W15=Winch; F,F0-F8=Floppy; Q=Quit
Select Drive :  W0[CR]
W0 Main Menu :  1)Parm 2)BadT 3)Form 4)Veri 5)Part 6)Writ P)Togl Q)Quit
     Command :  [CR]
```

```
 Winchester Drive 0 Menu:

        1) Display/Alter drive Parameters.
        2) Display/Alter Bad Track List.
        3) Format tracks.
        4) Verify tracks.
        5) Display/Alter VMEPROM Disk Partitions.
        6) Write out Header info to disk.
        P) Toggle Unit 2.
        Q) Quit & Select another Drive.
```

```
 W0 Main Menu:

        1)Parm 2)BadT 3)Form 4)Veri 5)Part 6)Writ P)Togl Q)Quit
        Command:  _
```

When dealing with a floppy drive, the display/alter commands do not allow you to alter the drive parameters, the bad track table, or the disk partitions, and you may not write out header information to a floppy disk.  To exit to VMEPROM, you must first return to the Select Drive Menu with the Q) command.  Following is a detailed description of the Drive Command Menu commands:

**1)  Display/Alter Drive Parameters:**

The Display/Alter Drive Parameters menu allows you to D)isplay the currently defined drive parameters, A)lter them, R)ead them in from a file, or Q)uit and exit to the Select Drive Menu:

```
W0 Parameters Menu :  A)lter, D)isplay, R)ead file, Q)uit
           Command :  _
```

To display the current drive parameters on a Winchester, enter the ‘D’ command.  The parameters are displayed to the screen.

The Drive Parameters that are displayed, and that can be altered are:

```
          Current (type) Drive N Parameters:
                # of Heads = Number of heads on drive
            # of Cylinders = Number of cylinders on drive
 Physical Blocks per Track = Actual blocks on a track
  Physical Bytes per Block = Actual bytes per physical block
          Shipping Cylinder = Where to position head before
                                 moving drive
                  Step rate = Controller dependent definition
 Reduced write current cyl = Cylinder to apply reduced
                                 write current
   Write Precompensate cyl = Cyl to apply write
                              precompensation
```

To alter them, enter the ‘A’ command. In the alter mode, you enter either: 1) a carriage return to leave the parameter the same and go to the next prompt; 2) a number and a carriage return to change the parameter and go to the previous parameter prompt. The Drive Parameters are displayed one at a time for you to either alter or leave alone.

If you have previously saved out the drive parameters to a disk file, you can restore them by entering the ‘R’ command, followed by the name of the file. This file may be created using the F)ile command of Drive Command Menu option 6) Write to disk, or it can be created with a VMEPROM editor. The format and syntax of the parameter file is discussed under option 6). Reading this information destroys all other information; replaces the parameters, the bad track table, and the partition definitions.

The ‘Q’ command returns you to the Drive Command Menu.

For example, look at floppy drive F0 parameters:

```
 Select Menu :  W,W0-W15=Winch; F,F0-F8=Floppy; Q=Quit
Select Drive :  F0[CR]
F0 Main Menu :  1)Parm 2)BadT 3)Form 4)Veri 5)Part 6)Writ P)Togl Q)Quit
     Command :  1[CR]
```

```
    Current Floppy Drive 0 Parameters:
                          # of Heads = 2
                      # of Cylinders = 80
           Physical Blocks per Track = 16
            Physical Bytes per Block = 256
                    Shipping Cylinder = 0
                            Step rate = 0
           Reduced write current cyl = 0
             Write Precompensate cyl = 0


       F0 Main Menu:
            1)Parm 2)BadT 3)Form 4)Veri 5)Part 6)Writ P)Togl Q)Quit
            Command: _
```

As another example, select the W0 Winchester and display the current parameters:

```
     W0 Main Menu:
            1)Parm 2)BadT 3)Form 4)Veri 5)Part 6)Writ P)Togl Q)Quit
          Command:  1[CR]
    W0 Parameters: A)lter, D)isplay, R)ead file, Q)uit
          Command:  D[CR]


    Current Winch Drive 0 Parameters:
                          # of Heads = 16
                      # of Cylinders = 1000
           Physical Blocks per Track = 32
            Physical Bytes per Block = 256
                    Shipping Cylinder = 0
                            Step rate = 0
           Reduced write current cyl = 0
             Write Precompensate cyl = 0

       W0 Parameters Menu:  A)lter, D)isplay, R)ead file, Q)uit
                  Command:  Q[CR]
             W0 Main Menu:  1)Parm 2)BadT 3)Form 4)Veri 5)Part 6)Writ
                            P)Togl Q)Quit
                  Command:  _
```

**2) Display/Alter Bad Track List:**

The Display/Alter Bad Track menu allows you to D)isplay the currently defined bad tracks on the drive (if any), add or delete tracks, C)lear the bad track table, get a H)elp message, or Q)uit and exit to the Drive Command Menu:

```
W0 Bad Tracks Menu:  Bad Track, D)isplay, C)lear, H)elp, Q)uit
             Command:  _
```

To display the current bad tracks on a Winchester, enter the 'D' command. The tracks are displayed on the screen in ascending order as a physical track number followed by the head and cylinder number, separated by a comma and enclosed in parentheses.

To add a bad track to the list, enter either the actual physical track number and a carriage return, or the head and cylinder number desired, separated by a comma and followed by a carriage return. To delete a track, precede the track or head number with a minus sign (-).

Sometimes the bad track table may be incorrect or spoiled. You can start all over again by entering the C)lear table command. The 'Q' command returns you to the Drive Command Menu. In case you have added or deleted some bad tracks, FRMT asks if you want to recalculate the disk partitions on the drive before returning to the drive menu. By altering the number of bad tracks, you also alter the number of logical tracks available for VMEPROM disk partitions. Answer 'Y' or 'N' to the query, as you like.

Note that the SCSI Winchesters handle bad blocks internally. So when you are using the ISCSI-1 controller, the bad blocks defined by the manufacturer are already spared on the disk.

For example, look at the Winchester drive 0 bad track list:

```
   W0 Main Menu:  1)Parm 2)BadT 3)Form 4)Veri 5)Part 6)Writ P)Togl

                  Q)Quit
        Command:  2[CR]
W0 Bad Tracks Menu:  Bad Track, D)isplay, C)lear, H)elp, Q)uit
        Command:  D[CR]

 Current Winch Drive 0 Bad Tracks:
231(0,77)    613(1,204)    697(1,232)    700(1,233)    703(1,234)

 W0 Bad Tracks Menu:  Bad Track, D)isplay, C)lear, H)elp, Q)uit
           Command:  Q[CR]
      W0 Main Menu:  1)Parm 2)BadT 3)Form 4)Veri 5)Part 6)Writ
                     P)Togl Q)Quit
           Command:  _
```

**3) Format Drive/Tracks:**

```
Sector Interleave = {default from MCONTB table is listed}
Physical Tracks to Format = {[CR] for beg,end tracks listed}
Ready to Format Drive 0 ? {'Y' or 'N'}
```

This routine first calls the INFMT routine which sets up the format. Then F)ormat makes one or more calls to the TKFMT routine until all the specified tracks are formatted.

Between calls, a check for user break ([CTRL-C]) is made, and the track number just formatted is printed to the terminal.

If there are errors, you can select either:

**R)etry, Y)es** -- add the track to the bad track list, or
**N)o** -- ignore the error and go on.

For example, format a floppy disk with the default sector interleave, 5, and do tracks 0 to 159, inclusive:

```
Sector Interleave = 5[CR]
Physical Tracks to Format = 0,159[CR]
Ready to FORMAT Floppy Drive 0 ? Y[CR]
Sector Interleave Table:
1,9,4,12,7,15,2,10,5,13,8,16,3,11,6,14

Issuing Format Drive Command

FORMAT Successful!
```

Note that the interleave is "Don’t care" for SCSI Winchester drives.

**4) Verify Tracks:**

Physical Tracks to Verify = {default from last format command}
Ready ? {‘Y’ or ‘N’}

This routine, after calling INFMT, reads every sector on each track specified. Errors are reported to the terminal. Between calls a check for user break ([CTRL-C]) is made, and the track just verified is printed to the terminal. If there are errors, you can select either R)etry, Y)es -- add the track to the bad track list, or N)o -- ignore the error and go on.

**5) Display/Alter Disk Partitions:**

The Display/Alter Partitions menu allows you to D)isplay the currently defined disk partitions, A)lter them, R)ecalculate them from the current values, or Q)uit and exit to the Drive Command Menu:

```
W0 Partitions Menu:  A)lter, D)isplay, R)ecalc, Q)uit
           Command:  _
```

To display the current disk partitions on a Winchester, enter the ‘D’ command. The partitions are displayed on the screen. The Disk Partitions that are displayed are based on a few parameters, which you can change:

```
       # of Large partitions = How many large divisions on the drive
      # of Floppy partitions = How many small divisions on the drive
First track for VMEPROM Parts = Where to begin the disk partitions
 Last track for VMEPROM Parts = Where to end the disk partitions
        First VMEPROM disk # = What is first VMEPROM disk # of
                               partitions
```

To alter them, enter the ‘A’ command. In the alter mode, you enter either: 1) a carriage return to leave the parameter the same and go to the next prompt; 2) a number and a carriage return to change the parameter and go to the next prompt; or 3) an escape to go to the previous parameter prompt. The disk partitions parameters are displayed one at a time for you to either alter or leave alone. If you alter the number of disks or the tracks for partitions, then you are asked if you would like to recalculate the partitions. Enter either ‘Y’ or ‘N’. If you only change the beginning VMEPROM disk number then only the disk numbers are reassigned, leaving the base and top tracks of the partitions alone.

You can make the partition information consistent by simply entering the ‘R’ command. This recalculates the drive partition information using the current values of drive parameters, bad track table, and partition parameters. The ‘Q’ command returns you to the Drive Command Menu.

```
  W0 Main Menu:  1)Parm 2)BadT 3)Form 4)Veri 5)Part 6)Writ P)Togl
                 Q)Quit
       Command:  5[CR]
W0 Partitions Menu:  A)lter, D)isplay, R)ecalc, Q)uit
       Command:  D[CR]

   Current Winch Drive 1 Disk Partitions:
       # of Large Partitions = 10
      # of Floppy Partitions = 12
First track for VMEPROM Parts = 0
 Last track for VMEPROM Parts = 15979
        First VMEPROM disk # = 2
  Total # of Logical Tracks = 16000
```

```
Disk #    Logical Trks  Physical Trks   VMEPROM sectors
          Base,Top      Base,Top        Total/{boot}
    2        0,1499        0,1500          47968/47776
    3        1500,2999     1501,3000       47968/47776
    4        3000,4499     3001,4500       47968/47776
    .        .             .               .
    .        .             .               .
    .        .             .               .
   24        15880,15959   15897,15979     2528/2336


W0 Partitions Menu:         A)lter, D)isplay, R)ecalc, Q)uit
     Command:               Q[CR]
      W0 Main Menu:         1)Parm 2)BadT 3)Form 4)Veri 5)Part 6)Writ
                            P)Togl Q)Quit
               Command:     _
```

**6) Write Header Information to Drive:**

The Write Header Information to Drive menu allows you to 1) ‘Y’ write the information to the drive header, 2) ‘N’ abort the command and return to the Drive Command Menu, or 3) ‘F’ write drive information to a file. After assigning the correct parameters to a drive, entering the bad tracks, formatting it, and partitioning it into VMEPROM disk numbers, you still need to write this information to the drive’s header. This information must reside on the disk and is used by for the BOOT ROMs and by VMEPROM.

This routine calls the INFMT subroutine to initialize the controller for the new number of heads and cylinders, and then calls the WTHED subroutine which writes out the drive data block (DDB) to the correct sector on the drive, usually sector 0.

To write this information to the drive, enter the ‘Y’ command. If you have second thoughts, enter the ‘N’ command.

You should save the information out to a floppy disk file for each Winchester drive. This file makes recovering from Winchester disasters easier. You can either select to only write out the file with the ‘F’ command, or write the file out after writing out the header information to the drive.

The file syntax is that:

1) lines starting with an asterisk (*) are ignored as comments;

2) parameter key words are four characters long and appear as the first four characters of the line;

3) key words are followed by an equal sign (=) and the value (hex must be preceded with dollar sign ($));

4) bad tracks use the key word TRACK, are followed by an equals sign, and are designated by either the track number or the head and cylinder numbers (separated by a comma);

5) order of the key words is not significant, except that the HEDS definition must precede any TRACK specification using the head, cylinder format; and

6) any unspecified key word parameters are reset to system defaults, and not left as previously entered values.

The drive parameter key words are defined as follows:

HEDS = # of Heads
CYLS = # of Cylinders
BPTK = Physical Blocks per Track
BPBK = Physical Bytes per Block
SHIP = Shipping Cylinder
STEP = Step Rate
REDU = Reduced Write Current Cylinder
WRTP = Write Precompensate Cylinder

The disk partition key words are defined as follows:

WPRT = # of Large Partitions
FPRT = # of Floppy Partitions
BTRK = First Track for VMEPROM Parts
ETRK = Last Track for VMEPROM Parts
BDKN = First VMEPROM disk #

While reading in the file using the R)ead command of the 1)Parameter menu, FRMT outputs a ‘Found:’ message, followed by the parameter value when a successful key word match and number conversion is made. This indicates that the parameter was loaded. If a key word match is not made or if the conversion fails, FRMT echoes the line to the terminal preceded by two question marks (??). This indicates that the parameter was not loaded.

Q) Select Another Drive:

If you were working with a floppy drive, the Q)uit command simply returns you to the Drive Select Menu. If you were working with a Winchester, then the Q)uit command asks whether or not to write the new drive data block down to low parameter RAM. Enter either ‘Y’ or ‘N’ to this query. If you answer ‘N’, your configuring session will be lost. It then exits to the Drive Select Menu.

```
W0 Main Menu:  1)Parm 2)BadT 3)Form 4)Veri 5)Part 6)Writ P)Togl
               Q)Quit
  Command:  Q[CR]

  Exit to Select Drive.  Update Param RAM (Y/N) ? Y[CR]
  System Parameter RAM Updated!!
  Select Menu: W,W0-W15=Winch; F,F0-F8=Floppy; Q=Quit
  Select Drive:  Q[CR]

?
```

### 1.2.34 FS - FILE SLOT USAGE

Format: FS

The FILE SLOT USAGE command lists all files currently open along with file slot information. When the first file is opened, it is assigned slot number 64; as successive files are opened, they are assigned file slots in numerical sequence down to 1. (Read Only Open allocates slots in the opposite order, from 1 to 32.) The file slot maintains information such as the current file pointers and sector indexes.

This data is defined as follows:

| | |
|---|---|
| Slot | File slot # |
| Name | File name;level/disk # |
| ST | File status |
| SM | Current sector in memory |
| PT | Current file pointer |
| SI | Sector index of SM |
| EOF | Sector index/# of bytes in END-OF-FILE sector |
| TN | Lock Task/Open Task |
| BF | Buffer pointer |
| FLGS | Lock flag/# Shared |

File status is defined as:

| | | | | |
|---|---|---|---|---|
| ST = | $8xxx | Sector altered | $xx80 | Altered |
| | $4xxx | File altered | $xx04 | Contiguous file |
| | $1xxx | Driver in channel | $xx02 | Delete protect |
| | $xAxx | Read only access | $xx01 | Write protect |
| | $x6xx | Shared random access | | |
| | $x2xx | Random access | | |
| | $x1xx | Sequential access | | |

Example:

```
? FS
Slot  Name        ST   SM   PT        SI   EOF      TN   BF        FLGS
64    fs1;101/6  C104 0142 00003916 0000 0000/82 0000 0000389E 00000000

? _
```

### 1.2.35 <u>GO - Start User Program</u>

```
Format: G
        G <address>
        GO
        GO <address>
```

A user program in memory is started with this command. The start address may be specified on the command line, or the value of the program counter, as displayed by the DR command, is taken if this field is omitted.

The following actions are taken by VMEPROM if this command is specified:

1) The processor registers are loaded with the user values.
2) The first instruction is executed.
3) If any breakpoints are defined, they are inserted in the user program.
4) The program is continued at the second instruction.

Example:

```
? G 8000
>>> This is a Test <<<

?
```

### 1.2.36 <u>GD - Start User Program Without Breakpoints</u>

```
Format: GD
        GD <address>
```

The GD command takes the same actions as the G or GO command, except that defined breakpoints are ignored and not inserted in the user program.

Example:

```
? GD 8000
>>> This is a Test <<<

?
```

### 1.2.37 GM - GET MEMORY

Format: GM
        GM <size>

The GM command adds memory to the current task. The amount of memory is specified by <size>. The <size> parameter has to be given in decimal. If no parameter follows GM, then all of the available memory is added. No error is reported if the memory request cannot be met.

Example:

```
? FM
No free memory contiguous to tasking memory

? FM 20
20 Kbytes free at address $00071800

? GM
? FM
No free memory contiguous to tasking memory

?
```

### 1.2.38 GOTO - GOTO String

Format: GOTO <string>

The GOTO command is used in procedure files to selectively process different commands. When the GOTO command is executed, the procedure file is rewound and all command line entries are ignored until a match is found with the <string> parameter and the command line. All preceding command lines to the match, including the matching command line, are ignored.

Execution continues with the next line.

The console echo flag (ECF$) is set to disable all console output until a match is found or the procedure file is exited. It is again restored after the label is found. Labels beginning with an asterisk are recommended since the monitor ignores them.

Example:

```
? TEST <cr>
? *START
? BT 100000 300000
? ER
Current error count = 0
? GOTO *START
```

### 1.2.39 GT - Start User Program with Temporary Breakpoint

```
Format:  GT <breakpoint>
         GT <breakpoint>,<address>
         GT <breakpoint>,<address>,<command>
         GT <breakpoint>,<address>,<command>,<count>
```

This is almost the same function as the G or GO command, except that an additional temporary breakpoint is inserted. This breakpoint is automatically removed if the program counter reaches this breakpoint.

If a command is given, it will be executed at the breakpoint. Otherwise all processor registers are displayed.

If a count is specified, the program first stops at the breakpoint when this specification has been achieved. The default value is one.

Example:

```
? GT 10020 10000
At temporary breakpoint
       0        1        2        3         4        5        6        7
D: 00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
A: 00000000 00000000 00000000 00000000  00000000 00001000 00007000 00099FFC

 VBR = 00000000   CAAR = 00000000   CACR = 00000001   SFC  = 0   DFC = 0
*USP = 00099FFC   SSP  = 00007BDE   MSP  = 000078C4
 PC  = 00010020   SR   = 0000 ..U..0........

? GT 10020 10000 lt

task   pri   tm  ev1/ev2  size     pc       tcb      eom      ports    name
*0/0    64    1            588  FF01FAB8 00007000 0009A000  1/1/0/0/0  lt

? GT 10020,10000,,2
At temporary breakpoint
       0        1        2        3         4        5        6        7
D: 00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
A: 00000000 00000000 00000000 00000000  00000000 00001000 00007000 00099FFC

 VBR = 00000000   CAAR = 00000000   CACR = 00000001   SFC  = 0   DFC = 0
*USP = 00099FFC   SSP  = 00007BDE   MSP  = 000078C4
 PC  = 00010020   SR   = 0000 ..U..0........

? _
```

### 1.2.40 <u>HELP - HELP</u>

Format: HELP
        HELP <command>

The HELP command first displays a short description of all VMEPROM built-in commands on the terminal. Then a more detailed description of all commands is displayed.

After every screen full, the output stops.  It may be continued by entering a <cr>.  Control is transferred back to the command interpreter on any key other than <cr>.

If HELP is followed by a command name, a short description of this command is displayed.

If HELP is followed by one or more characters, but not a complete command name, a start description of all commands matching with the given character is displayed.

Example:

```
? HE M
M <address>[,B|W|L&N&O|E]      Modify memory contents
MD <address>[,<count>]            Display memory in Hex and
ASCII
MF <filename>                  Make file
MM                             Alias for M command
MS <address>,<data|"string">  Preset memory with constant or
                               string

? _
```

### 1.2.41 <u>HIST - Command history</u>

Format: HIST

The HIST command is used to show which commands can be recalled with [CTRL-A]. This is an easy way to check if a command is inside the alternate command line buffer. If it is, recalling the line is possible and it need not to be written a second time.

Example:

```
? HIST
BT 10000 20000
DR
BT 200000 300000

? [CTRL-A]
BT 10000 20000[CTRL-A]
DR[CTRL-A]
BT 200000 300000<cr>

? _
```

### 1.2.42 IA - IF ALTERED

Format: IA <file name>.<command>

The IF ALTERED command tests and clears the altered file bit of the directory entry specified by <file name>. If the file had the alter bit set (indicated in the directory listing by a '+' under type), then execution of the command line continues. Otherwise, the rest of the line is ignored.

This command is useful in assembly procedures to update object modules when many files are involved and only a few may have changed.

Example:

```
? IA test.DT
? DT
16-Mar-88
16:47:38

? IA test.DT
?
```

### 1.2.43 ID - SET SYSTEM DATE/TIME

Format: ID

The SET SYSTEM DATE/TIME command displays the VMEPROM header and prompts for the date and time. The header shows the version of VMEPROM and the used CPU-type as displayed after reset.

The date can be entered in either a day, ASCII month, year form or numeric month, day, year.

Any delimiter can be used to separate date and time parameters.
Pressing [CR] leaves the old date and time.

Example:

```
? ID
*****************************************************************
*                                                               *
*                        V M E P R O M                          *
*                                                               *
*           SYS68K/CPU-xx     Version  a.bb      Date           *
*                                                               *
*            (c) FORCE Computers  and  Eyring Research          *
*                                                               *
*****************************************************************

Date: 17-Aug-89   <cr>
Time: 18:45:21    <cr>

? _
```

### 1.2.44 **IN - Install an utility**

Format: IN
IN [-Saddress|-Mmemory,]<name>[,<arguments>]
IN <-name>

The IN command is used to install utilities memory resident. Access to this utilities is much faster as if they have to be loaded from a winchester or floppy.

The first format is used to display all installed utilities, the third is necessary to uninstall a utility.

The second format is needed to install a utility. If the utility is already resident in RAM, ROM etc. the start address should be passed as first parameter preceeded by <-S>. In this case VMEPROM will not allocate memory for this utility. For utilities loaded from disk additional memory can be allocated. Therefore the first parameter has to be the number of Kbytes of additional memory preceeded by <-M>. VMEPROM will now allocate memory for the utility plus the additional memory as required. Also command line arguments may be passed to the utility, if they are needed for installation.

Example:

util1 is memory resident at $80000.
uitl2 is on disk and its size is about 12 Kbytes. 20 Kbytes of memory are needed for gobal variables.

```
? IN -S800000,util1
Utility util1 installed

? IN -M20,util2
Utility util2 installed

? IN
UTIL.     NAME        BEGINADDRESS
1         util1       $800000          Memory resident
2         util2       $aa000           Size = 32 Kbyte

? _
```

### 1.2.45 INFO - Information about the CPU board

Format: INFO

The INFO command is used to obtain information about the CPU board. The output is strongly dependent on the used CPU board.

These outputs are given at all CPU types:

1) CPU type.

2) VMEPROM Version and it's start address.

3) EPROM base address.

4) I/O devices: Depending on the CPU type all I/O devices are listed including their base address.

5) RAM addresses: SYRAM start address.
   : Current tasks task control block start address.

Additionally some information will occur, depending on the CPU board.

Example:

```
? INFO
FORCE CPU - xx
VMEPROM Version a.bb at $FF000008

EPROM base addresses:
  System EPROM    at $FF000000

I/O Devices:
  IHDL            at $FF805800;   RTC             at $FF803000
  PI/T 1          at $FF800C00;   PI/T 2          at $FF800E00
  MFP             at $FF805000;   DUSC1 channel A at $FF802000
  DUSC1 channel B at $FF802020;   SCSI            at $FF803400

RAM addresses:
  Local RAM    $0 to $000FFFFF
  SYRAM           at $00001000;   TCB             at $0009D000

? _
```

### 1.2.46 INIT - Initialize a Disk for Use with VMEPROM

Format: INIT [<disk>[,<directory size>[,<disk size>[,<disk name>]]]]

The INIT command initializes a floppy or Winchester for the usage with VMEPROM. The disk must be formatted (see FRMT command).

The required parameters are:

1. disk number
2. number of directory entries
3. physical size of the disk in number of 256-byte sectors
4. disk name

All parameters may be specified on the command line or may be entered interactively after the function has been invoked. If interactive input is used, default values are given. All given values can then be edited. The number of sectors shows the total number of formatted sectors on the specified disk. This number can be edited or the string "MAX" or "BOOT" can be entered. "MAX" will show the total number of formatted sectors (default value); "BOOT" shows the maximum number of sectors on a disk with exception to free space which is reserved for a bootstrap. The disk name which is given shows the actual name of the specified disk. If none is shown, the disk has not been named. Of course, the given name can be edited.

If interactive input is used, a final confirmation has to be entered. Otherwise no confirmation has to be given.

Typical values for INIT are:

```
    for floppies:    128  directory entries
                    2528  sectors

 for Winchesters:   512 or 1024 directory entries
                    dependant on the specification with FRMT,
                    option 5.
                    (See FRMT command for details.)
```

All values are expected to be given in decimal.

Example:

```
? INIT 9,128,2528,Diskname
Init:  Disk # 9
       Directory entries: 128
       Number of sectors: 2528
       Disk name: Diskname
Initializing...

? INIT
Enter Disk # : 2<cr>
Directory Entries : 1024<cr>
Number of sectors 1..47968, BOOT, MAX: 47968<cr>
Disk Name : Test-Disk<cr>
```

**(Example cont'd)**

```
Init: Disk # 2
      Directory entries: 1024
      Number of sectors: 47968
      Disk name: Test-Disk
Initialize disk named 'Test-Disk' ?  y<cr>
Initializing...

? _
```

### 1.2.47 INSTALL - INSTALL UARTS OR DISK DRIVER

```
FORMAT: INSTALL [?]
        INSTALL -<U<type>|W<number>>
        INSTALL U<uart type>,<address | filename>[,board base
                address]
        INSTALL W,<address | filename>[,board base address]
                [,number of desired disks(F/W)|<P>partition
                list][,partition offset]
```

The INSTALL command installs, lists or removes device codes (disk drivers, UARTs). If there is no parameter given, all installed UARTs and disk drivers are shown. If the first parameter is equal to a question mark, all UARTs and disk drivers, which are already in EPROM, are listed.

INSTALL UARTs:

VMEPROM can handle up to eight UART types. Each type has a table of short branches (DSR table) for various subroutines to get, put, baud etc. If a certain UART type is not used in the system, a "NE" status is returned for all calls. To install a new UART type, set the first parameter to U1, U2 up to Un, where n is the number of UART types in the system. If the number is out of range, then an error appears. The ability to pick a type is not currently used in the system. This means that uninstall must first be used with this UART type by preceding the first parameter with a minus sign.
The second parameter can have the filename of the DSR object code or the base address where the object code starts. In case of a filename being written, the INSTALL facility first loads the object code into memory and preserves that memory. It then calls the initialization routine for the card and enters a jump table for this UART into a global jump table for UARTs. In this case, the UART type ALSO reserved a small RAM area of maximum 64 bytes.

The optional third parameter, <board base address>, is the base address of the first card of the new type, as jumpered in the system. The DSR table has the same entries as the standard PDOS UART type, with the following additions. The data word just after the DSR table must contain the characters "U0"(Uzero), the word just after that must have a BRA.S INIT branch to the card initialization routine. The INSTALL assumes that after the initialize call that there is a string, null terminated, which describes the UART type.

If VMEPROM finds the magic word $A557 there after, an uninstall will be supported.

Each UART entry is defined as follows:

```
UDSR    BRA.S   UDG              ;GET CHARACTER
        BRA.S   UDP              ;PUT CHARACTER
        BRA.S   UDB              ;BAUD UART
        BRA.S   UDR              ;RESET UART
        BRA.S   UDS              ;READ UART STATUS
        BRA.S   UHW              ;HIGH WATER
        BRA.S   ULW              ;LOW WATER
        DC.B    'U0'             ;UART ID
        BRA.S   UDI              ;INSTALL
UNAME   DS.B    'NAME',0         ;NAME OF DRIVER (ZERO TERMINATED)
        EVEN
        DC.W    $A557            ;MAGIC
        DC.W    P_TYP            ;PROCESSOR TYP
        BRA.W   UNINS            ;UNINSTALL

        P_TYP  = %000000000000xxxx
                              ||||___ 68000
                              |||____ 68010
                              ||_____ 68020
                              |______ 68030
```

The INIT call is made by INSTALL in supervisor mode. This routine has the following inputs and outputs:

```
UDI - INSTALL DRIVER
      IN: A1.L = K1$BEGN
          A2.L = OPTIONAL CARD BASE ADDRESS OR ZERO
          A5.L = SYRAM BASE
          A6.L = BEGIN OF TCB
          (A7)  = RETURN ADDRESS
         4(A7)  = RAM ADDRESS IN GLOBAL DSR TABLE
     OUT: D0.W = -1 ERROR
                 NUMBER OF CARDS
```

VMEPROM also supports an uninstall routine with following inputs:

```
UNINS - UNINSTALL DRIVER
        IN: (A7) = RETURN ADDRESS
           4(A7) = RAM ADDRESS IN DSRTAB
```

INSTALL DISK DRIVER:

VMEPROM handles four disk drivers linked in a driver list. To install a new disk driver set the first parameter to W. If the device code is resident in memory or EPROM, the second assignment <address|filename> should be the driver start address. If a filename is given, the INSTALL facility first allocates memory and loads the object code into memory. INSTALL calls the initialization routine (INIT) for the disk controller to enter the new disk driver into the driver list.  If there are already four disk drivers installed an error occurs and a driver is first uninstalled by setting the first parameter to -Wn. n is the disk driver number given, when you call INSTALL without parameters.  The third to fifth assignment are optional parameters.

The <base address> is the base address of the card as jumpered in system. The fourth parameter <number of desired disks | <P>partition list> allows you to select only one or more physical disks (FLOPPY/HARD DISKS) or by preceding a P to select one or more logical disks. If no fourth parameter is given the driver will handle all disks are found (maximum 2 FLOPPY DISKS and 4 HARD DISKS for each driver). The fifth parameter <partition offset> is an offset added to all logical partition numbers for that driver. Each installable disk file must have a specific structure on top of file that helps INSTALL to handle them. There are two structures handled by VMEPROM. If there is any write protect for the object code of the disk driver (i.e. the code is in EPROM), the driver file must have the following structure:

```
WBEG    DC.W    'W0'    ; IDENTIFIER
        BRA.S   INIT    ; INIT DISK
        BRA.L   XDOF    ; DISK OFF
        NOP
        NOP
        NOP
        BRA.L   XREAD   ; READ SECTOR
        NOP
        NOP
        NOP
        BRA.L   XWRIT   ; WRITE SECTOR
        NOP
        NOP
        NOP
        DC.B    'WSAMPLE',0
        EVEN
```

If there is no write protect the driver file can also have the following structure (like as used by PDOS), and VMEPROM will overwrite all BSR with a BRA.

```
WBEG    DC.W    'W0'    ; IDENTIFIER
        BRA.S   INIT    ; INIT DISK
        BSR.L   XDOF    ; DISK OFF
        JMP     $0.L    ; OLD DISK OFF ROUTINE
*                       ; (PROVIDE ADDRESS AT INSTALL TIME)
        BSR.L   XREAD   ; READ SECTOR
        JMP     $0.L
        BSR.L   XWRIT   ; WRITE SECTOR
        JMP     $0.L
        DC.B    'WSAMPLE',0
        EVEN
```

The driver file always starts with an identifier "W0" and after the little jump table INSTALL assumes a string, null terminated, which describes the driver.

The initialization routine has the following inputs and outputs:

```
INIT  -  INSTALL DISK DRIVER

          IN: A1.L = K1$BEGN
              A2.L = OPTIONAL CARD BASE ADDRESS
              D7.W = OPTIONAL DISKNR (BY VMEPROM SET TO FFFF)
         OUT: DO.W = -1 ERROR
                     NUMBER OF CARDS
```

NOTE: The UART for the I/O devices on-board of the CPU card are installed by default, but a disk driver is only installed by default if set by the front panel switches.

Example:

```
? INSTALL ?

    THE FOLLOWING UARTS AND DISK DRIVER ARE ALREADY IN EPROM:

    UART    ONBOARD_20                ADDR: $FF005000
    UART    FORCE SIO-1/2             ADDR: $FF005400
    UART    FORCE ISIO-1/2            ADDR: $FF005800
    DISK    FORCE ISCSI-1             ADDR: $FF005C00
    DISK    FORCE WFC-1               ADDR: $FF006400

? INSTALL

THE FOLLOWING DRIVERS ARE INSTALLED:

UART   NAME        BEGINADDRESS   PROCESSOR
U1     ONBOARD_20  $FF005000      68020/30

THERE ARE NO DISK/DRIVERS INSTALLED

? INSTALL W,80C500,,P3/4/9-11,30
  DISK DRIVER FORCE ISCSI-1 INSTALLED

? INSTALL

THE FOLLOWING DRIVERS ARE INSTALLED:

UART   NAME           BEGINADDRESS   PROCESSOR
U1     ONBOARD_20     $FF005000      68020/30

DISK   NAME           BEGINADDRESS   F/W   FIRST DISK(W) PHYSICAL DISK
DRV0   FORCE ISCSI-1 $FF005C00       2/1   33            F0,F1,W0-W3
```

### 1.2.48 KM - KILL MESSAGE

Format: KM
        KM <task #>

The KM command removes all task messages associated with <task #> from the message buffers.

If no task is specified, then all messages associated with the current task are deleted from the message buffers.

See also 1.2.69 SM - SEND MESSAGE.

### 1.2.49 KT - KILL TASK

Format: KT
        KT {-}<task #>

The KILL TASK command removes a task from the task list and returns the task's memory to the free pool for use by other tasks. Only your current task or a task spawned by your task can be killed. (Task 0 can kill any task except itself or a task that is kill protected.)

Each task is assigned a unique task number which is shown by the LIST TASK command. Only the current task (indicated by '*') or those spawned by the current task (indicated by current task number following a "/" character) may be killed. Task #0 is the system task and cannot be killed.

If a minus sign (-) precedes the task number, then the task's memory is not deallocated to the memory bit map. If the task number is zero, then the current task is killed without deallocating memory.

If no parameter is given, then the current task is killed and memory is deallocated.

All open files associated with the killed task are closed by the KT command.

Example:

```
? LT
task  pri  tm  ev1/ev2  size     pc       tcb      eom      ports    name
*0/0   64  10            600  FF01FAB8 00007000 0009D000  1/1/0/0/0  LT
 1/0   64   2       98   100  FF002986 0009D000 000B6000  2/2/0/0/0

? KT 1

? LT
task  pri  tm  ev1/ev2  size     pc       tcb      eom      ports    name
*0/0   64  10            600  FF01FAB8 00007000 0009D000  1/1/0/0/0  LT

? _
```

### 1.2.50 <u>LC - LIST DIRECTORY</u>

Format: LC <file list>

The LIST DIRECTORY command displays a selected list of disk file names. The file names are printed in a compressed format with 5 names on every line.

The files are selectively listed according to file name, extension, level, disk number, file attribute, or date of last change.

The format of the <file list> is defined as follows:

```
   <file list> = {file}{:ext}{;level}{/disk}{/select...}
where:  {file} = 1 to 8 characters (1st alpha) (@=all,*=wild)
        {:ext} = 1 to 3 characters (:@=all,*=wild)
      {;level} = directory level (;@=all)
       {/disk} = disk number ranging from 0 to 255 (/@=all)
     {/select} = /AC = Assign Console file
                 /BN = Binary file
                 /BX = VMEPROM BASIC token file
                 /EX = VMEPROM BASIC file
                 /OB = 68000 VMEPROM object file
                 /SY = System file
                 /TX = Text file
                 /DR = System I/O driver
                 /*  = Delete protected
                 /** = Delete and write protected
                 /Fdy-mon-yr = selects files with date of
                               last change greater than
                               or equal to 'dy-mon-yr'.
                               /Fmn/dy/yr format can also
                               be used.
                 /Tdy-mon-yr = selects files with date of
                               last change less than or
                               equal to 'dy-mon-yr'.
                               /Tmn/dy/yr format can also
                               be used.
```

In the file list specification, the '@' character indicates all subsequent characters match and the '*' character is a single character wild card.

Also displayed is the disk name, number of files stored on disk and number of directory entries available. This information is useful in disk maintenance. The directory entries are not necessarily in alphabetical order but in the order they are stored in the disk directory.

See also: 1.2.53 LS - List directory sequential

Example:

```
? LC
  test          lv            ls            lc
Number of files: 4            Sectors allocated:  5
?
```

### 1.2.51 <u>LD - LOAD FILE</u>

```
Format: LD <file name>
        LD <file name>,<start address>
```

The LOAD FILE command loads a file into memory but does not begin executing it. The file must be of the type 'SY'. The starting load address is optionally specified by <start address>. Otherwise it defaults to immediately following the TCB.

This command can be used to debug files, load multiple files or to load programs outside of known tasking memory.

The LOAD FILE command uses the XLDF primitive and loads 'SY' files four bytes at a time. As a result, as many as three extra bytes may be loaded.

Example:

```
? ld test1,8000
? di 8000 5
8000   NOP
8002   NOP
8004   NOP
8006   NOP
8008   NOP

?
```

### 1.2.52 LO - Load S-record

Format: LO
        LO <address> , <command line>,<-V|-T>

The LO command loads a S-record into memory from a standard input port. Normal I/O redirection may be used for input from other ports. The starting load address is optionally specified by <address>.

An optional command line may be specified which is sent to the host before S-record loading starts. It can be used to initiate a host system download without using the TM Command.

Two possible options exist which must be proceeded by a minus sign. If option V is given, the contents of the S-records will only be compared with contents of those memory locations which are to be loaded. The different values of the memory locations and the S-record data are displayed. If option T is given without an address parameter, the S-records are loaded immediately following the TCB. The following S-record types are supported by VMEPROM:

S0 Start record, ignored by VMEPROM and may be omitted.

S1 Data record with 16 bit address field

S2 Data record with 24 bit address field

S3 Data record with 32 bit address field

S7 End record with 32 bit address field

S8 End record with 24 bit address field

S9 End record with 16 bit address field

If the address for the LO command is specified on the command line, address fields in the data records are ignored and the S-record is loaded contiguously from the specified address upwards.

If the end record address field is equal, 0 control is transferred back to the VMEPROM command interpreter. If the address file holds an address, VMEPROM automatically executes a "G address" command after the S-record is loaded and an end record is found. Because of the "G" command all breakpoints which are defined are inserted in the program.

See also: 1.2.27 DU - Dump S-records

Example:

```
? lo <2 8800
?
```

## 1.2.53 LS - LIST DIRECTORY

Format: LS <file list>

The LIST DIRECTORY command displays a selected list of disk file names. The file listing also includes the directory level, file type, file size, start sector address, date of creation, and date of last update.

The files are selectively listed according to file name, extension, level, disk number, file attribute, or date of last change.

The format of the <file list> is defined as follows:

```
<file list> = {file}{:ext}{;level}{/disk}{/select...}
where:    {file} = 1 to 8 characters (1st alpha)
(@=all,*=wild)              {:ext} = 1 to 3 characters
(:@=all,*=wild)
      {;level} = directory level (;@=all)
       {/disk} = disk number ranging from 0 to 255 (/@=all)
     {/select} = /AC = Assign Console file
                 /BN = Binary file
                 /BX = VMEPROM BASIC token file
                 /EX = VMEPROM BASIC file
                 /OB = 68000 VMEPROM object file
                 /SY = System file
                 /TX = Text file
                 /DR = System I/O driver
                 /*  = Delete protected
                 /** = Delete and write protected
                 /Fdy-mon-yr = selects files with date of
                               last change greater than
                               or equal to 'dy-mon-yr'.
                 /Fmn/dy/yr format can also be used.
                 /Tdy-mon-yr = selects files with date of
                               last change less than or
                               equal to 'dy-mon-yr'.
                 /Tmn/dy/yr format can also be used.
```

In the file list specification, the '@' character indicates all subsequent characters match and the '*' character is a single character wild card.

Also displayed with each directory listing is the disk name, the number of files stored on the disk and the number of directory entries available.

This information is useful in disk maintenance.

The directory entries are not necessarily in alphabetical order but in the order they are stored in the disk directory.

See also: 1.2.50 LC - List Directory

Example:

```
? LS
Lev  Name:ext    Type      Size     Sect    Date created    Last update
102  test          C          1     013B  00:50 16-Mar-88 00:51 16-Mar-88
102  lv           +C          1     0145  00:56 16-Mar-88 00:56 16-Mar-88
102  ls            C          1     0146  00:56 16-Mar-88 00:56 16-Mar-88
Number of files: 3            Sectors allocated:  3

?
```

### 1.2.54 LT - LIST TASKS

Format: LT

The LT command displays all tasks currently in the task list to the console. Task 0 is the system task and is created automatically during system initialization. This task cannot be killed.

Your current task is indicated by an '*' preceding the task number. Following the task number is a slash and the parent task number. Subsequent data provides the current status of each task and is defined as follows:

| | |
|---|---|
| task | {*=current} Task #/parent task # |
| pri | Task priority (1-255) |
| tm | Time slice (1-255) |
| ev1/ev2 | Suspended event(s) |
| size | Task size (k bytes) |
| pc | Current program counter<br>If the task is in suspended state or ready state the program counter points to the first opcode this task will execute after the task is moved to run state. |
| tcb | Task control block |
| eom | End of memory |
| ports | Task I/O ports in the following order:<br>input port/output port/Unit 2 port/Unit 4 port/Unit 8 port |
| name | The name of the command currently executing |

Example:

```
? LT
task  pri  tm  ev1/ev2  size     pc       tcb      eom      ports    name
*0/0   64  10            600  FF01FAB8 00007000 0009D000  1/1/0/0/0  LT

? _
```

### 1.2.55 LV - DIRECTORY LEVEL

Format: LV
        LV <level>

The DIRECTORY LEVEL command displays or sets the current directory level used in directory listings and file definitions.

The DIRECTORY LEVEL command without any argument displays the current directory level. A file defined without a specified directory level is defined on the current level.

If an argument is specified, it is converted to a number and sets the current directory level.

The range is from 0 to 255 in decimal.

The disk directory is soft partitioned into 256 different groups, facilitating file maintenance. A soft partition means that any file is accessible from any current level. It also means that file names must be unique for each disk number (disk directory).

Example:

```
? LV
Level = 103

? LV 100
? LV
Level = 100

?
```

### 1.2.56 M - Modify Memory

Format:

```
M <address>[,<option>]
MM <address>[,<option>]
```

Option is B | W | L & N & O | E

The Modify Memory command is used to inspect and change memory locations. Several options are allowed on the command line to specify the size of the memory and the access type. The following options are allowed:

B  memory is byte sized (8 bits).
W  memory is word sized (16 bits). This is the default.
L  memory is long word sized (32 bits).
O  memory is byte sized and on odd addresses only.
E  memory is byte sized and on even addresses only.
N  memory is write only, the current contents is not displayed.

The Odd and Even options are overriding the B/W/L options. The N (no read) option has to be specified after the size qualifier and after the Odd/Even specification. All memory accesses check that the write access was successful by performing a read after the write unless N is specified. If the data written and the data read do not match, the command is terminated and an error message is displayed.

The memory modify command supports a number of sub-commands, which can be entered instead of a new memory value. These sub-commands do not change the access option specified on the command line.

The following sub-commands are supported:

```
<cr>          open next location
 =            open same location again
 -            open previous location
 -<count>     go back <count> bytes
 +            open next location
 +<count>     go forward <count> bytes
 #<address>   open new absolute address
 ?<mnemonic>  insert 68000 opcode at current address
 .            exit to the command interpreter
```

Example:

```
? M 8000
8000   4246 : <cr>
8002   1C2E : <cr>
8004   0441 : <cr>
8006   4247 : ?nop<cr>
8008   A05A : -2<cr>
8006   4E71 : -<cr>
8004   0441 : #8000<cr>
8000   4246 : <cr>
8002   1C2E : .
?
```

### 1.2.57 MD - Display Memory

Format: MD <address>
        MD <address>[,<count>]

The MD command displays the memory contents of the specified address. The data is displayed in hex and ASCII representation, 16 bytes on every line. If the hex value cannot be displayed in ASCII representation, a full stop (".") is displayed instead.

If no count is specified on the command line, the Display Memory command displays 16 lines, representing 256 bytes of data, and prompts the user to display more or to return to the command interpreter.

If a carriage return (<cr>) is entered, the next 256 bytes are displayed. Any other character returns control back to the command interpreter of VMEPROM.

If a count is specified on the command line, the value is interpreted as the number of bytes to be displayed. All values are assumed to be in hex.

If a base is specified with the BASE command this value is printed at the first line which is put out.

Example:

```
? MD 8000 30
00008000:  A0 0E 00 00 00 21 00 08  01 00 00 00 00 1C 00 04  .....!..........
00008010:  00 00 00 00 00 00 00 00  00 00 00 08 00 00 00 00  ................
00008020:  40 B0 00 00 24 E4 00 04  02 D5 00 00 00 80 00 08  @...$...........

? MD A000 30
0000A000:  08 98 00 00 04 88 00 01  00 80 00 08 40 08 00 80  ............@...
0000A010:  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0000A020:  40 03 00 00 00 00 00 00  02 04 00 40 00 00 00 00  @.........@....

? BASE 2000

? MD 8000 30
00002000+
00008000:  08 98 00 00 04 88 00 01  00 80 00 08 40 08 00 80  ............@...
00008010:  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00008020:  40 03 00 00 00 00 00 00  02 04 00 40 00 00 00 00  @.........@....

? _
```

### 1.2.58 <u>MF - MAKE FILE</u>

Format: MF <file>

The MF command allows an ASCII file to be created from the user console. The <file> must be previously defined or preceded by a ’#’. The normal line editing is permitted but once a return key has been entered, the line is written to the file.

A [CTRL-C] cancels the line without writing it to the file. An [ESC] terminates the process, closes the file, and returns to the VMEPROM monitor.

The MF command uses the XGLB primitive and hence, normal editing control characters are available and lines are limited to 78 characters. Control characters other than those used for editing cannot be entered (i.e. this includes a TAB character.)

Example:

```
? MF test
This is a test file to show the<cr>
functionality<cr>
of<cr>
the MF command.<cr>
<esc>

? SF test
This is a test file to show the
functionality
of
the MF command.

?
```

### 1.2.59 MS - Set Memory to Constant or String

Format: MS <address>,<data|"string">

This command writes the specified data pattern to memory. The data may consist of hex numbers and ASCII data in any combination. The ASCII data must be put in inverted commas.

Example:

```
 ? BF 8000 8100 @377 B

 ? MD 8000 20
 00008000:  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
 00008010:  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................

 ? MS 8000 "Hello World"0D0A00

 ? MD 8000 20
 00008000:  48 65 6C 6C 6F 20 57 6F  72 6C 64 0D 0A 00 FF FF  Hello World.....
 00008010:  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................

 ? _
```

### 1.2.60 PROMPT - CHANGE PROMPT SIGN

Format: PROMPT [<data|"string">]

The PROMPT command is used to change the prompt for the current task in the used specified pattern.

The data may consist of hex numbers and ASCII data in any combination. The ASCII data must be put in inverted commas.

If no parameter is given, the default VMEPROM prompt "?" will occur. The user defined prompt sign will be truncated to nine characters maximum.

Example:

```
? PROMPT "#"_
#_

#PROMPT ("HELLO> ")_
HELLO> _

HELLO> PROMPT_
? _
```

### 1.2.61 <u>RC - RESET CONSOLE</u>

Format: RC

The RESET CONSOLE command is used in an Assigned Console (type=AC) file to terminate the procedure and to revert back to the system console. This allows for a graceful termination of the file commands by closing the file and prompting for a new command.

Since procedure files can be nested, only the current procedure file is closed.

### 1.2.62 **RD - RAM DISK**

```
Format:  RD
         RD {-}<unit>[,<size>][,<address>]
```

The RAM DISK command sets/displays the current RAM disk's units, sizes and memory addresses. VMEPROM maintains a RAM disk list, providing up to 4 RAM disks at any time. Each RAM disk has a disk number and separate memory address. The RAM disk command allows you to add, delete, renumber and list RAM disks. When the address or size is changed, the RAM disk must again be initialized. This is done by preceding the RAM disk unit by a minus sign. Otherwise, the INIT command can be used to initialize the disk.

The default RAM disk setup of VMEPROM is described in the User's Manual of your CPU - board. If there is no parameter the RAM disks are listed showing disk number, size number in sectors and base address. They are not in a defined order.

The first assignment <unit> specifies the disk number to be used for the RAM disk. It must be in the range of 0-99.

The argument <size> specifies the size of the RAM disk in sectors. Each sector has a size of 256 bytes. The given size will be rounded up to 2 Kbyte boundary. A RAM disk of 32 Kbytes will have 128 sectors. If the second parameter <size> is zero, the RAM disk <unit> is removed from the list. To aid memory management, if the <unit> was positive or zero, the memory used by that RAM disk is deallocated in free memory pool for new tasks or other RAM disks. If <unit> negative, memory is not deallocated. If the second parameter <size> is non zero, either a new RAM disk is entered into the list or an existing RAM disk is renumbered.

If there is no third assignment <address>, a new RAM disk is made of <size> sectors coming from either the free memory pool, if possible, or from the calling task's memory. If there is a third parameter <address>, then VMEPROM seeks <address> among currently defined RAM disks. If there is a match, the new <unit> and <size> replaces those of the current disk at <address>. (no check is made that <size> is the same.) If there is no matching address, the new RAM disk is entered in the list, and no memory management is performed.

Example:

```
? RD
Ram disk unit = 8, size = 128, address = $00077DFC

? RD -50,100,$800000
? RD
Ram disk unit = 8,  size = 128, address = $00077FDC
Ram disk unit = 50, size = 104, address = $00800000
```

### 1.2.63 RM - Modify Processor Registers

Format: RM
RM <register>
RM <register>,<value>

The RM command modifies the processor registers or, if available, the data registers of the 68881 coprocessor. Three modes are allowed.

The first mode is an interactive mode, which scans all registers. For each register, the current value is displayed and the user is prompted to enter a new value. A <cr> leaves the register unchanged. After a new value or a <cr> has been entered, the same procedure will be started for the next register. If an <ESC> or <.> has been entered, control is transferred back to the command interpreter.

The second mode makes it possible to change only one specified register. The current value is then displayed and the user is prompted to enter a new value. A <cr> leaves the register unchanged. After a new value or a <cr> has been entered, control is transferred back to the command interpreter.

The third mode allows the specification of the new value for the given register on the command line and does not display the the old value.

The following registers may be modified by the user:

| | |
|---|---|
| VBR | Vector base register, only on 68010/68020/68030 systems |
| SFC/DFC | Source and Destination function code register |
| CAAR | CACHE address register, only for 68020/68030 systems |
| CACR | CACHE control register, only for 68020/68030 systems |
| PC | Program counter |
| SR | Status register |
| USP | User Stack pointer |
| SSP | System Stack pointer |
| MSP | Master Stack pointer, only on 68020/68030 systems |
| D0-D7 | Data registers D0-D7 |
| A0-A7 | Address registers A0-A7, where A7 is the current stack pointer as defined by the status register |
| FP0-FP7 | Floating point Coprocessor registers, if available. |

Caution: Be careful when modifying the Vector Base register (VBR) as VMEPROM is a interrupt driven system and any modifications to this register may crash the system.

Example:

```
? RM D0
D0 = 00000000  : 12345678<cr>

? RM D1 1000

? DR
       0        1        2        3         4        5        6        7
D: 12345678 00001000 00000000 00000000  00000000 00000000 00000000 00000000
A: 00000000 00000000 00000000 00000000  00000000 00001000 00007000 0009CFFC

 VBR = 00000000   CAAR = 00000000   CACR = 00000001   SFC  = 0   DFC = 0
*USP = 0009CFFC   SSP  = 00007BE6   MSP  = 000078C4
 PC  = 00008000   SR   = 0000 ..U..0........

? RM FP0
FP0 =  0.00000000 E+000  : 1234.56E-24<cr>

? DRF
FP0: 1.23456000 E-021  0.00000000 E+000  0.00000000 E+000  0.00000000 E+000
FP4: 0.00000000 E+000  0.00000000 E+000  0.00000000 E+000  0.00000000 E+000

? _
```

### 1.2.64 RN - RENAME FILE

```
Format: RN <file1>,<file2>
        RN <file1>,<level>
        RN <file1>,<file2>[,<file3>,<level>[,...]]
```

The RENAME FILE command changes the file name stored in the disk file directory. The RENAME command may also be used to move a file from one directory level to another. The file <file1> is renamed to <file2>. A disk specification in the second parameter is meaningless. If a number <level> is used instead of <file2>, the <file1> is moved to the new level.

The third format is used to rename multiple files. Each pair of arguments (<file1>/<file2>,<file3>/<level>) is treated as standing alone.

Example:

```
? lc
  temp          rn1
Number of files: 2            Sectors allocated:  2

? rn temp,temp1
? lc
  temp1         rn1
Number of files: 2            Sectors allocated:  2

?
```

### 1.2.65 RR2 - EPROM Programming

```
Format: RR2 [<f>,<file>],<board>,<mode>,<option>
        RR2 [<m>,<addr>,<cnt>],<board>,<m>,<opt>
```

The RR2 command is used for programming EPROMS or EPROMS on a SYS68K/RR-2/RR-3 board. It can also be used to transfer files or memory contents into a SRAM area on the RR_2 or to load EPROM/EEPROM contents into the VMEPROM memory.

The following are examples on the usage of the RR2 command:

```
    ? RR2 F,FILENAME,RR_2_ADDRESS,MODE,OPTION
      if the source is a disk file, or

    ? RR2 M,STRTADDR,BYTECNT,RR_2_ADDRESS,MODE,OPTION
      if the source is in memory.
```

The following describes the parameters:

```
F,FILENAME.............source = disk file
                            F = source flag
                     FILENAME = the name of the source file
M,STRTADDR,BYTECNT.....source = memory
                            M = source flag
                     STRTADDR = source start address
                      BYTECNT = source length in bytes
RR_2_ADDR..............the address of the RR_2 bank
MODE...................1 = 8 bit mode (single EPROM)
                       2 = 16 bit mode (two EPROMS)
                       4 = 32 bit mode (four EPROMS)
OPTION.................P = program an EPROM (includes E and
V
                                            and a bit test)
                       E = check if EPROM is empty
                       V = verify source and EPROM contents
                       L = load EPROM contents to memory
```

For further information on the hardware setup of the SYS68K/RR2 or SYS68K/RR3 board please refer to the user’s manual of the RR-2/3 board.

Example:

```
? RR2 M,$0,$8000,$800000,2,E
```

executes an empty check in word mode for EPROM type 27128 (16k x 8) at RR_2 address $800000. The M - source flag and the memory address are dummy.

```
? RR2 F,PROG/2,$800000,4,P
```

programs EPROMS at address $800000 in 32-bit mode with the source file PROG from disk 2.

? RR2 M,$10000,$2000,$800000,1,L
loads the contents of an 8k x 8 EPROM at address $800000 into the memory to address $10000.

SYS68K/RR-2/RR-3 board configuration:

This example contains the RR-2 board configuration and and the program usage for 27128 EPROMs in the 16 bit mode.

Jumper settings for 16k x 8 EPROMs on bank 2 (TOSHIBA 27128):

```
   B1b          Read time selection on bank 2

                8     5
                o o o o
                  | |                              250 ns
                o o o o
                1     4

   B2b          Write time selection on bank 2

                3       15
                o o o o o
                    |
                o o o o o                           50 ms

                o o o o o
                1       13

   B4b          Device type bank 2

                4
                o o
                  |                                 EPROM type
1
                o o
                1

   B13b         Device size bank 2

                10      6
                o o o o o
                | | | |                          4 x 16k x 8
                o o o o o
                1       5


   B15          Device pinning bank 2

                3                   33
                o o o o o o o o o o o
                |       |
                o o o o o o o o o o o
                    |           | |
                o o o o o o o o o o o
                1                   31
```

This page left intentionally blank

B16 Enable VPP generator

```
2
o
|
o
1
```

B17 Select VPP bank 2

```
3
o
|
o                                21V

o
1
```

B18 Select output enable on VPP bank 2

```
2
o

o
1
```

B19 Select chip erase bank 2

```
3
o

o
|
o
1
```

B11 Upper address bank 2

```
2     8
o o o o
  | | |                          $8
o o o o
1     7
```

B12 Lower address bank 2

```
2     8
o o o o
| | | |                          $0
o o o o
1     7
```

Program call for subsequent jobs:

a) EPROM empty check

```
? RR2 M,$0,$8000,$800000,2,E
```

- option = empty check
- mode = word
- RR-2 base address
- byte count (2 EPROMs 16k x 8)
- memory address (don’t care)
- source = memory

b) program EPROMs

```
? RR2 F,MYFILE:PRG/4,$800000,2,P
```

- option = program
- mode = word
- RR-2 base address
- source file name
- source = file

c) load EPROMs into memory

```
? RR2 M,$10000,$8000,$800000,2,L
```

- option = load
- mode = word
- RR-2 base address
- byte count(2 EPROMs 16k x 8
- memory address
- destination = memory

### 1.2.66 RS - RESET DISK

Format: RS
        RS <disk #>

Disk files must be closed at the end of any task so that sector buffers are flushed to the disk, pointers updated in disk directories, and file slots released for further usage. The RESET command either closes all open files associated with your task or closes all open files on a specified disk. The first mode allows your task to terminate itself without affecting the files of other tasks, while the second mode is used before withdrawing a disk from a disk drive.

RESET also clears the assigned console FILE ID (ACI$(A6)).

However, the assigned console file may not be closed if the RESET disk option is used and the file resides on a different disk.

Example:

```
? FS
Slot  Name       ST   SM     PT       SI   EOF     TN   BF       FLGS
64    fs1;101/6  C104 0142 00003916 0000 0000/82 0000 0000389E 00000000
? RS
? FS
Slot  Name       ST   SM     PT       SI   EOF     TN   BF       FLGS

?
```

### 1.2.67 SA - SET FILE ATTRIBUTES

Format: SA <file>
        SA <file>,<attributes>

The SET FILE ATTRIBUTES command associates file attributes with a file in the disk directory. File attributes include file types and protection flags.

The following file types are supported by VMEPROM:

| | |
|---|---|
| AC | Assign Console, command file. |
| SY | 680x0 executable file, memory image. |
| TX | Text file. |
| DR | Loadable driver. |
| C | Contiguous file. This type can not be set or reset by the user. |
| * | Delete protected. Allowed in addition to other types. |
| ** | Delete and write protected. Allowed in addition to other types. |

Note: The file type "C" is an addition to the other file types and is set by VMEPROM. It cannot be set or cleared by the user.

The following types are not decoded or used by VMEPROM but may be used:

| | |
|---|---|
| BN | Binary file. |
| EX | Basic file. |
| BX | Basic file. |
| OB | VMEPROM object file. |

The following gives a detailed description of all file types

supported by VMEPROM:

1. AC - Assign console. A file typed 'AC' specifies to VMEPROM that all subsequent requests for a console character will be intercepted and the character obtained from the assigned file.

2. SY - System file. A 'SY' file is generated automatically by the Save File to Memory command. Also the LD (Load file to memory) command checks for the SY type. If any program shall be loaded from disk to memory and be executed, it must have the type SY.

3. TX - ASCII text file. A 'TX' type classifies a file as containing ASCII character text.

4. DR - System I/O driver. A 'DR' file type is a VMEPROM system I/O driver. Channel buffer data is treated as a program and is used to extend the file system to I/O devices. The Loadable I/O drivers are described in detail in the Appendix.

5. * - Delete protect. The file is delete protected and cannot be deleted from the disk. This file type is an addition to the other file types.

6. ** - Delete and write protect. The file cannot be deleted or written to by any system call. This file type is an addition to the other file types.

Example:

```
? SA FILE       Clears all attributes (except ’C’)
? SA FILE,SY    Sets SY type only
? SA FILE,**    Sets protection only
? SA FILE,OB**  Sets type and protection
```

```
 ? LS
   Lev  Name:ext Type  Size  Sect  Date created     Last update
   101  temp1    +C       1  0110  19:47 16-Mar-88  19:47 16-Mar-88

Number of files: 1            Sectors allocated:  1

? SA temp1 TX
? LS
Lev  Name:ext  Type   Size    Sect  Date created      Last update
101  temp1     TX+C     1     0110  19:47 16-Mar-88   19:47 16-Mar-88

Number of files: 1            Sectors allocated:  1

?
```

### 1.2.68 <u>SF - SHOW FILE</u>

Format: SF {-}<file name>

The SHOW FILE command displays the disk file as specified by <file name> on your console. The output is paged and truncated to 78 characters per line unless the file name is preceded with a minus sign. Pressing [ESC] terminates the command at any time.

If a minus sign precedes the file name, then the file is displayed without line truncations or paging. Again, [ESC] terminates the command.

Example:

```
? SF test
This is a test file to show the
functionality
of
the MF command.

?
```

### 1.2.69 <u>SM - SEND MESSAGE</u>

Format:

```
        SM <task #>,<message>
```

The SEND MESSAGE command puts an ASCII text message in a message buffer. The destination is specified by <task#>. The message can be up to 63 characters in length.

If a message is sent to itself, i.e. the task which is sending the message, the complete message is interrupted as a command line and executed.

Note: No other commands can be appended to an ’SM’ command with a period, since the <message> parameter takes everything up to the carriage return.

See also: 1.2.48 KM - KILL MESSAGE.

Example:

```
? SM 2,Hallo
? SM 0, LV
? LV
Level=1
?
```

### 1.2.70 SP - DISK SPACE

```
Format: SP
        SP <disk #>
```

The DISK SPACE command displays the current number of defined files, the total possible directory size, the number of disk sectors free, the largest possible contiguous file, the number of disk sectors used, and the number of allocated disk sectors.
All numbers represent decimal sectors. The total size in bytes is the number of sectors times 252.

The <disk #> specifies the disk number. If no parameter is used, then the default disk is displayed.

The 'Files' parameter lists the current number of defined files in the disk directory. This is followed by the maximum number of files definable in the directory.

The 'Free' parameter shows the number of sectors still available for file storage. This is followed by the largest number of contiguous sectors. This is helpful in defining contiguous files.

The 'Used' parameter shows exactly how much of the disk is truly used versus the amount of disk storage allocated. Some files may have END-OF-FILE markers pointing within the file and not at the end. If these files were copied to another disk, the unused storage would be recovered.

Example:

```
? SP 6
Files= 16/128
Free = 2080/1596
Used = 288/293

?
```

### 1.2.71 <u>ST - SET TASK TERMINAL TYPE</u>

```
Format: ST
        ST <type>
```

The ST command sets the position cursor (PSC$) and clear screen (CSC$) variables in the task control block (TCB). This command makes it easy to use various types of terminals together with VMEPROM. Each task has its own characters for these two functions, which are initialized, when the task is started, to the parent task control set.

If a legal <type> is passed in the command line, then ST simply enters the corresponding sequences into the user status block.

Otherwise, the command prints the following table of options:

```
      D = VT52
      L = Lear Siegler ADM3a
      V = VT100
      T = TVI 950
      U = User defined
   Type = _
```

and prompts the user for an input. Enter the letter representing the type of terminal you are using.

The terminal type setup is only required for the VMEPROM screen editor. No other function uses the terminal dependant sequences.

The default setup of VMEPROM is the codes for a VT52 terminal.

In addition to the built in terminal types, the ST command allows to enter the values for position cursor, clear screen, clear to end of screen and clear to end of line interactively with the "C" option. So nearly every terminal can be used with VMEPROM.

```
?  St U  to to enter a user defined terminal

Enter encoded position cursor value: $.
```

Now the position cursor code can be entered in hex. The hex value must be 16 bit. The format of the leading characters for cursor positioning is as follows (note that each letter represents a bit):

```
B111 1111 0222 2222

B = 0 then $00 bias
    1 then $20 bias
O = 0 then row before column, 1 then column before row
1 = 7 bits for first ASCII lead in character
2 = 7 bits for second ASCII lead in character
A value of 0 will result in the code for a VT100 terminal.

Enter encoded clear screen value: $_
```

The cursor home and clear screen can also be entered as a encoded 16 bit value. The format is (note that each letter represents a bit):

```
  E111 1111 E222 2222

  E = if 1 then precede with [ESC]
  1 = 7 bits for first ASCII character
  2 = 7 bits for second ASCII character
  If all 16 bits are 0 then a VT100 is selected
```

Enter encoded clear to end of screen value: $.

This is the code to clear the access from the current cursor position to end of screen. The format is:

```
  0111 1111 0222 2222

  1 = 7 bit for first ASCII character
  2 = 7 bit for second ASCII character
```

Enter encoded clear to end of line value: $_

This is the code to clear from the cursor position to the end of the line. The format is:

```
0111 1111 0222 2222

1 = 7 bit for first ASCII character
2 = 7 bit for second ASCII character
```

Example:

```
? ST
   D = VT52
   L = Lear Siegler ADM3a
   V = VT100
   T = TVI 950
   U = User defined
Type = L

? ST D
?
```

### 1.2.72 <u>SV - Save Memory to File</u>

Format:  SV <begin>,<end>,<filename>

The SAVE TO FILE command writes binary memory images to the file specified by <file>.  The parameters <begin> and <end> specify the start and end memory bounds.

The file is created on the disk if it does not exist. The file gets the file type ’SY’.

Example:

```
? SV 8000 8100 file
?
```

### 1.2.73 <u>SY - SYSTEM DISK</u>

```
Format: SY
        SY <disk1>{,<disk2>{,<disk3>{,<disk4>}}}
```

The disk device identifier is contained within the file name. However, a default or system disks are assigned by the SY command.
On all open and define commands, file names without the disk identifier follow the system disk specification order in looking for the file on disk. All other commands use only the first system disk specification.

Example:

```
? SY
System disks : 6

? SY 6,2
? SY
System disks : 6,2

?
```

### 1.2.74 <u>T - Trace Program Execution</u>

```
Format: T
        T <address>[,<begin>,<end>]
        T <R|S|?>
        TT
        TT <address>[,<begin>,<end>]
        TT <R|S|?>
```

The first format starts a user program in trace mode. The start address is the current value of the program counter (PC) as displayed by the DR command.

The second format is used to start a user program in trace mode at the specified address. Additionally two parameters (<begin> and <end>) are able to be given. These parameters specify an address range. Inside this range the program does not stop tracing.

The third format is used to display/set the trace mode.
The parameter "S" toggles between enabling and disabling trace over subroutine. No stop inside a subroutine (i.e. started with BSR) will be done if trace over subroutine is enabled.
The parameter "R" toggles between displaying the registers after each step and displaying only if trace count matches or the condition for trace over range is true. Displaying registers goes along with displaying the disassembled code of the next instruction which will be executed.
The parameter "?" induces the displayal of the current settings.

If the program stops the user is prompted to continue the trace or to return to VMEPROM. Tracing can be continued by entering a space (" ") or a carriage return (<cr>).

See also: 1.2.75 TC - Set Trace Count
1.2.77 TJ - Trace on change of flow

Example:

```
? DI 8000 7
8000   SUBA.L A5,A5
8002   ADDQ.L #1,A5
8004   BSR.B $800A
8006   ADDQ.L #3,A5
8008   XEXT
800A   ADDQ.L #2,A5
800C   RTS

? DR T
PC = 00008000  SP = 003B67FC  A6 = 00007000  A5 = 00001000

? T ?
Display registers after each step
Trace over subroutine is disabled
```

**(Example cont'd)**

```
? T 8000
Trace
PC = 00008002  SP = 003B67FC  A6 = 00007000  A5 = 00000000
8002 : ADDQ.L #1,A5<cr>
Trace
PC = 00008004  SP = 003B67FC  A6 = 00007000  A5 = 00000001
8004 : BSR.B $800A<cr>
Trace
PC = 0000800A  SP = 003B67F8  A6 = 00007000  A5 = 00000001
800A : ADDQ.L #2,A5<cr>
Trace
PC = 0000800C  SP = 003B67F8  A6 = 00007000  A5 = 00000003
800C : RTS<cr>
Trace
PC = 00008006  SP = 003B67FC  A6 = 00007000  A5 = 00000003
8006 : ADDQ.L #3,A5<cr>
Trace
PC = 00008008  SP = 003B67FC  A6 = 00007000  A5 = 00000006
8008 : XEXT<cr>

? T 8000 800A 9000
Trace
PC = 00008002  SP = 003B67FC  A6 = 00007000  A5 = 00000000
8002 : ADDQ.L #1,A5<cr>
Trace
PC = 00008004  SP = 003B67FC  A6 = 00007000  A5 = 00000001
8004 : BSR.B $800A<cr>
Trace
PC = 0000800A  SP = 003B67F8  A6 = 00007000  A5 = 00000001
800A : ADDQ.L #2,A5                                 NO STOP!
Trace
PC = 0000800C  SP = 003B67F8  A6 = 00007000  A5 = 00000003
800C : RTS                                          NO STOP!
Trace
PC = 00008006  SP = 003B67FC  A6 = 00007000  A5 = 00000003
8006 : ADDQ.L #3,A5<cr>
Trace
PC = 00008008  SP = 003B67FC  A6 = 00007000  A5 = 00000006
8008 : XEXT<cr>

? T R
Display registers only if stop condition reached

? T 8000 800A 9000
Trace
PC = 00008002  SP = 003B67FC  A6 = 00007000  A5 = 00000000
8002 : ADDQ.L #1,A5<cr>
Trace
PC = 00008004  SP = 003B67FC  A6 = 00007000  A5 = 00000001
8004 : BSR.B $800A<cr>
Trace
PC = 00008006  SP = 003B67FC  A6 = 00007000  A5 = 00000003
8006 : ADDQ.L #3,A5<cr>
Trace
PC = 00008008  SP = 003B67FC  A6 = 00007000  A5 = 00000006
8008 : XEXT<cr>
```

**(Example cont'd)**

```
? T S
Trace over subroutine is enabled

? T 8000
Trace
PC = 00008002  SP = 003B67FC  A6 = 00007000  A5 = 00000000
8002 : ADDQ.L #1,A5<cr>
Trace
PC = 00008004  SP = 003B67FC  A6 = 00007000  A5 = 00000001
8004 : BSR.B $800A<cr>
Trace
PC = 00008006  SP = 003B67FC  A6 = 00007000  A5 = 00000003
8006 : ADDQ.L #3,A5<cr>
Trace
PC = 00008008  SP = 003B67FC  A6 = 00007000  A5 = 00000006
8008 : XEXT<cr>

? _
```

### 1.2.75 TC - Set Trace Count

Format: TC <count>

The Set Trace Count command sets the number of instructions to be traced continuously. The default after reset is 1.

See also: 1.2.74 T - Trace program execution
1.2.77 TJ - Trace on change of flow

Example:

```
? TC
Trace count = 0

? TC 100
? TC
Trace count = 100

?
```

### 1.2.76 TIME - Enable/Disable Display of the Program Run Time

```
Format: TIME
        TIME ON
        TIME OFF
```

VMEPROM has the ability to measure the run time of user programs or command execution of the built in commands. This feature can be turned on and off with the TIME command. If only TIME is entered, the current status is displayed (i.e. On or OFF). VMEPROM displays the time in minutes, seconds, and tens and hundreds of seconds. If time measurement is enabled, a time stamp is taken whenever the command interpreter gets a complete input line. The timing stops when the function is executed and control is transferred back to the command interpreter.

Example:

```
? TIME
Time is off

? TIME ON
? BENCH 1 8000
Bench  1: Decrement long word in memory, 10.000.000 times
Benchmark time = 0:07.23
Programm execution time is 0:07.27

? TIME OFF
?
```

### 1.2.77 TJ - Trace on Change of Flow

Format: TJ
        TJ <address>

This command is only supported on 68020 versions. It traces a user program (like the Trace command), but only on instructions where a change of program flow occurs. Such instructions are for example: BRA, BSR, JMP, JSR, RTS etc.

See the Trace command for a complete description of program tracing.

See also: 1.2.74 T - Trace program execution

**Note:** This command is only available for 32 bit processors.

### 1.2.78 <u>TM - TRANSPARENT MODE</u>

Format: TM <port #>
        TM <port #>,<break>

The TRANSPARENT MODE command directs your current input to <port #>. Input received from <port #> is directed to your output. This command effectively allows you to access other systems as if you were a terminal.

This process continues until an [ESC] character is entered. This can be changed to another character by adding the <break> parameter.

Caution: Typing ^C twice will abort every command currently in the state of execution. This is independent of the brake character.

### 1.2.79 TP - TASK PRIORITY

Format: TP
        TP <task #>
        TP <task #>,<[time * 256 +] priority>

The TASK PRIORITY command allows you to change task priority of different tasks. The task number is specified by <task #> and defaults to the current task if omitted. If no priority is given the tasks current priority is displayed. Otherwise the tasks priority is changed to the given value.

The range of <priority> is from 1 to 255 where 255 is the highest priority. The highest priority, ready task always executes. Tasks on the same priority level are scheduled in a round robin fashion. The time a task is in running state is also given with the <priority> parameter. If no time is specified the time slice will not be changed. Otherwise it is calculated to "time*256+priority".

Example:

```
? LT
task  pri  tm  ev1/ev2  size   pc       tcb      eom      ports   name
 0/0  64   10       97  600  FF002986 00007000 0009D000 1/1/0/0/0
*1/0  63   2            100  FF01FAB8 0009D000 000B6000 2/2/0/0/0 LT

? TP
Current tasks priority = 63, time slice = 2

? TP 0
Task #0 priority = 64, time slice = 10

? TP 1,100

? TP 1
Task #1 priority = 100, time slice = 2

? LT
task  pri  tm  ev1/ev2  size    pc       tcb      eom     ports   name
 0/0  64   10       97  600  FF002986 00007000 0009D000 1/1/0/0/0
*1/0  100   2           100  FF01FAB8 0009D000 000B6000 2/2/0/0/0 LT

? TP 1, $1064

? TP 1
Task #1 priority = 100, time slice = 16

? _
```

### 1.2.80 UN - CONSOLE UNIT

```
Format: UN
        UN {[-128]-}<unit number>
```

The CONSOLE UNIT command displays/sets the console output unit number. Unit 1 is the system terminal. Unit 2 and 3 are auxiliary output ports. The unit 4 is used by VMEPROM for output redirection and shall not be used.

The first format is used to display the current output unit number.

The second format selects where the output is to be directed. If the parameter is negative no character echo to the input port will be done. Otherwise character echo to the input port is enabled.
If the parameter is lower than -128 only the system prompt will be displayed at the input port. No character echo of the input port is done. The correct parameter for this command is calculated to "-128-unit number". This command is very helpful to recognize if a command line can be entered.

Example:

```
? UN
Unit mask = 1

? UN 3

? UN
Unit mask = 3

? UN -1
{LT}                               No echo!

task pri tm  ev1/ev2  size     pc       tcb      eom     ports    name
*0/0 64  1            700  FF01FAB8 00007000 000B6000  1/1/0/0/0  LT
{UN -129}                          No echo

? {LT}                             No echo
task pri tm  ev1/ev2  size     pc       tcb      eom     ports    name
*0/0 64  1            700  FF01FAB8 00007000 000B6000  1/1/0/0/0  LT

? {UN 1}                           No echo

? _
```

### 1.2.81 <u>ZM - ZERO MEMORY</u>

Format: ZM

The ZERO MEMORY command clears the entire user work space to zeros. All flags and pointers are reset.

The memory is cleared from the end of the TCB up to the current user stack pointer. The values on the stack are not destroyed.

Example:

```
? ZM
?
```

# VMEPROM SYSTEM CALLS

T A B L E   O F   C O N T E N T S

**T A B L E  O F  C O N T E N T S (cont'd)**

**T A B L E   O F   C O N T E N T S (cont'd)**

# 1. VMEPROM SYSTEM CALLS

## 1.1 General Information

PDOS assembly primitives are assembly language system calls to PDOS. They consist of one word A-line instructions (words with the first four bits equal to hexadecimal ’A’). PDOS calls return results in the 68000 status register as well as regular user registers.

PDOS calls are divided into three categories:

1) system
2) console I/O
3) file support primitives.

The following primitives, which are available in a standard PDOS operating system environment are not available in VMEPROM:

XBUG Calls the PDOS debugger, this module is not included in VMEPROM
XCHF PDOS monitor command, not included in VMEPROM
XLST PDOS monitor command, not included in VMEPROM
XBFL PDOS monitor command, not included in VMEPROM
XAIM PDOS monitor command, not included in VMEPROM
XGTP PDOS monitor command, not included in VMEPROM
XEXZ PDOS monitor command, not included in VMEPROM

These primitives give reference to the PDOS Monitor or PDOS Debugger and these modules are not included in VMEPROM.

The monitor calls XGNP and XPCB of PDOS are emulated by VMEPROM and perform their expected functions.

## 1.2 Assembly Language Calls

PDOS assembly primitives are one word A-line instructions which use the exception vector at memory location $00000028. Most primitives use 68000 registers to pass parameters to and results from resident PDOS routines. Observe the following example for Trapping an error after a PDOS call:

```
        CALLX   LEA.L   FILEN(PC),A1 ;GET FILE NAME
                XSOP                 ;OPEN FILE, ERROR?
                BNE.S ERROR          ;Y
                MOVE.W  D1,SLTN(A4)  ;N, SAVE SLOT #
```

PDOS primitives return error conditions in the processor status register. This provides error processing allowing a program to do long or short branches on different error conditions.

PDOS command primitives can be grouped into six levels according to their function and calling hierarchy. These levels are System Calls, System Support Calls, Console I/O Calls, File Support Calls, File Management Calls, and Disk Access Calls.

Level 1 PDOS primitives consist of system calls that deal with functions such as swapping, message passing, events, TRAP vector initialization, etc.

The PDOS system calls are as follows:

| | |
|---|---|
| **XGML** - Get memory limits | **XTEF** - Test event flag |
| **XGUM** - Get user memory | **XDEV** - Delay set/reset event |
| **XFUM** - Free user memory | **XSUI** - Suspend until interrupt |
| **XRTS** - Read task status | **XDTV** - Define trap vectors |
| **XSTP** - Set/read task priority | **XSUP** - Enter supervisor mode |
| **XLKT** - Lock task | **XUSP** - Return to user mode |
| **XULT** - Unlock task | **XRSR** - Read status register |
| **XSWP** - Swap to next task | **XLSR** - Load status register |
| **XCTB** - Create task block | **XRTE** - Return from interrupt |
| **XKTB** - Kill task | **X881** - 68881 enable |
| **XSTM** - Send task message | **XDMP** - Dump memory from stack |
| **XGTM** - Get task message | **XRDM** - Dump registers |
| **XKTM** - Kill task message | **XEXC** - Execute PDOS call D7.W |
| **XGMP** - Get message pointer | **XLER** - Load error register |
| **XSMP** - Send message pointer | **XERR** - Return error D0 to VMEPROM |
| **XSEV** - Set event flag | **XEXT** - Exit to VMEPROM |
| **XSEF** - Set event flag w/swap | **XEXZ** - Exit to VMEPROM with command |

Level 2 consists of system support calls. Data conversion and data/time processing are their main functions. They are as follows:

**XCBD** - Convert binary to decimal
**XCBH** - Convert binary to hex
**XCBM** - Convert to decimal w/message
**XCDB** - Convert decimal to binary
**XCBX** - Convert to decimal in buffer
**XCHX** - Convert binary to hex in buffer
**XRDT** - Read date
**XRTM** - Read time
**XRTP** - Read time parameters
**XFTD** - Fix time & date
**XPAD** - Pack ASCII date
**XUAD** - Unpack ASCII Date
**XUDT** - Unpack date
**XUTM** - Unpack time
**XWDT** - Write date
**XWTM** - Write time
**XGNP** - Get next parameter

Level 3 primitives deal with console I/O. Included are commands for setting the baud rate and other characteristics of an I/O port, reading and writing characters or lines, clearing the screen, positioning the cursor, and monitoring port status.

**XGCB** - Conditional get character
**XGCC** - Get character conditional
**XGCR** - Get character
**XGCP** - Get port character
**XGLB** - Get line in buffer
**XGLM** - Get line in monitor buffer
**XGLU** - Get line in user buffer
**XPBC** - Put buffer to console
**XPCC** - Put character(s) to console
**XPCL** - Put CRLF
**XPCR** - Put character raw
**XPSP** - Put space to console
**XPLC** - Put line to console
**XPDC** - Put data to console
**XPEL** - Put encoded line to console
**XPMC** - Put message to console
**XPEM** - Put encoded message to console
**XCLS** - Clear screen
**XPSC** - Position cursor
**XTAB** - Tab to column
**XRCP** - Read port cursor position
**XBCP** - Baud console port
**XSPF** - Set port flag
**XRPS** - Read port status
**XCBC** - Check for break character
**XCBP** - Check for break or pause

Level 4 primitives are file support calls for the file manager. However, important functions such as copying files, appending files, sizing disks, and resetting disks are included here.

**XFFN** - Fix file name
**XLFN** - Look for name in file slots
**XBFL** - Build file directory list
**XRDE** - Read next directory entry
**XRDN** - Read directory entry by name
**XAPF** - Append file
**XCPY** - Copy file
**XLDF** - Load file
**XRCN** - Reset console inputs
**XRST** - Reset disk
**XSZF** - Get disk size

Level 5 primitives are the file management calls of PDOS. They use the file lock (event 120) to prevent conflicts between multiple tasks. Functions such as defining, deleting, reading, writing, positioning, and locking are supported by the file manager.

**XDFL** - Define file
**XRNF** - Rename file
**XRFA** - Read file attributes
**XWFA** - Write file attributes
**XWFP** - Write file parameters
**XDLF** - Delete file
**XZFL** - Zero file
**XSOP** - Open sequential
**XROO** - Open random read only
**XROP** - Open random
**XNOP** - Open non-exclusive random
**XLKF** - Lock file
**XULF** - Unlock file
**XRFP** - Read file position
**XRWF** - Rewind file
**XPSF** - Position file
**XRBF** - Read bytes from file
**XRLF** - Read line from file
**XWBF** - Write bytes to file
**XWLF** - Write line to file
**XFBF** - Flush buffers
**XFAC** - File altered check
**XCFA** - Close file w/attribute
**XCLF** - Close file

The final level of primitives is for disk access via the read/write logical sector routines in the PDOS BIOS. A disk lock (event 121) is used to make these calls autonomous and prevent multiple commands from being sent to the controller.

**XISE** - Initialize sector
**XRSE** - Read sector
**XWSE** - Write sector
**XRSZ** - Read sector zero

## 1.3 Description of Kernel Primitives

This chapter gives a detailed description of all Kernel calls which are available in VMEPROM.

### 1.3.1 X881 - SAVE 68881 ENABLE

Mnemonic: X881
Value: $A006
Module: MPDOSK1
Format: X881

Description: The SAVE 68881 ENABLE sets the BIOS save flag (SVF$(A6)) thus signaling the PDOS BIOS to save and restore 68881 registers and status during context switches. The save flag is again cleared by exiting to VMEPROM.

See also: None

Possible Errors: None

## 1.3.2 XAPF - APPEND FILE

Mnemonic: XAPF
Value: $A0AA
Module: MPDOSF
Format: XAPF
<status error return>

Registers: In (A1) = Source file name
(A2) = Destination file name

Note: A [CTRL-C] will terminate this primitive and return error -1 in data register D0.

Description: The APPEND FILE primitive is used to append two files together.

The source and destination file names are pointed to by address registers A1 and A2, respectively. The source file is appended to the end of the destination file. The source file is not altered.

See also: None

Possible Errors:

-1 = Break
50 = Invalid file name
53 = File not defined
60 = File space full
61 = File already open
68 = Not PDOS disk
69 = Not enough file slots
Disk errors

### 1.3.3 XBCP - BAUD CONSOLE PORT

```
Mnemonic:      XBCP
Value:         $A070
Module:        MPDOSK2
Format:        XBCP
               <status error return>

Registers: In   D2.W = f0PI 8DBS / <port #>
                D3.W = Baud rate
                D1.W = Port type
                D5.L = Port base
```

Description: The BAUD CONSOLE PORT primitive initializes any one of the PDOS I/O ports and binds a physical UART to a character buffer. The primitive sets handshaking protocol, receiver and transmitter baud rates, and enables receiver interrupts.

Data register D2 selects the port number and sets (or clears) the corresponding flag bits. If D2.W is negative, then the absolute value is subsequently used and the port number is stored in U2P$(A6).

The right byte of data register D2 (bits 0-7) selects the console port.

The left byte of D2.W (bits 8-15) selects various flag options including ^S-^Q and/or DTR handshaking, receiver parity and interrupt enable, and 8-bit character I/O.

The receiver and transmitter baud rates are initialized to the same value according to register D3. Register D3 ranges from 0 to 7 or the corresponding baud rates of 19200, 9600, 4800, 2400, 1200, 600, 300, or 110.

If data register D4 is non-zero, then it selects the port type and register D5 selects the port base address. These parameters are system-defined and correspond to the UART module. If register D4 is zero, there is no change.

See also: 1.3.78 XRPS - READ PORT STATUS
1.3.92 XSPF - SET PORT FLAG

Possible Errors:

66 = Invalid port or baud rate

### 1.3.4 XCBC - CHECK FOR BREAK CHARACTER

```
Mnemonic:      XCBC
Value:         $A072
Module:        MPDOSK2
Format:        XCBC
               <status return>

Registers:     Out  SR = EQ....No break
                         LO....[CTRL-C], Clear flag & buffer
                         LT....[ESC], Clear flag
                         MI....[CTRL-C] or [ESC]
```

Note: If the ignore control character bit ($02) of the port flag is set, then XCBC always returns .EQ. status.

Description: The CHECK FOR BREAK CHARACTER primitive checks the current user input port break flag (BRKF.(A5)) to see if a break character has been entered. The PDOS break characters are [CTRL-C] and the [ESC] key. A [CTRL-C] sets the port break flag to one, while an [ESC] character sets the flag to a minus one. The XCBC primitive samples and clears this flag. The condition of the break flag is returned in the status register. An 'LO' condition indicates a [CTRL-C] has been entered. The break flag and the input buffer are cleared. All subsequent characters entered after the [CTRL-C] and before the XCBC call are dropped.

All open procedure files are closed and any system frames are restored. Also, the last error number flag (LEN$) is set to -1 and a '^C' is output to the port. An 'LT' condition indicates an [ESC] character has been entered. Only the break flag is cleared and not the input buffer. Thus, the [ESC] character remains in the buffer.

The [CTRL-C] character is interpreted as a hard break and is used to terminate command operations. The [ESC] character is a soft break and remains in the input buffer, even though the break flag is cleared by the XCBC primitive. (This allows an editor to use the [ESC] key for special functions or command termination.)

Note: If the ignore control character bit ($02) of the port flag is set, then XCBC always returns .EQ. status.

See also: None

Possible Errors: None

### 1.3.5 XCBD - CONVERT BINARY TO DECIMAL

Mnemonic: XCBD
Value: $A050
Module: MPDOSK3
Format: XCBD

Registers: In D1.L = Number
Out (A1) = String

Description: The CONVERT BINARY TO DECIMAL primitive converts a 32-bit, 2's complement number to a character string. The number to be converted is passed to XCBD in data register D1. Address register A1 is returned with a pointer to the converted character string located in the monitor work buffer (MWB$).

Leading zeros are suppressed and a negative sign is the first character for negative numbers. The string is delimited by a null. The string has a maximum length of 11 characters and ranges from -2147483648 to 2147483647.

See also: 1.3.9 XCBX - CONVERT TO DECIMAL IN BUFFER.

Possible Errors: None

### 1.3.6 XCBH - CONVERT BINARY TO HEX

Mnemonic: XCBH
Value: $A052
Module: MPDOSK3
Format: XCBH

Registers: In D1.L = Number
Out (A1) = String

Description: The CONVERT BINARY TO HEX primitive converts a 32-bit number to its hexadecimal (base 16) representation. The number is passed in data register D1 and a pointer to the ASCII string is returned in address register A1. The converted string is found in the monitor work buffer (MWB$) of the task control block and consists of eight hexadecimal characters followed by a null.

See also: 1.3.12 XCHX - CONVERT BINARY TO HEX IN BUFFER.

Possible Errors: None

### 1.3.7 XCBM - CONVERT TO DECIMAL W/MESSAGE

Mnemonic: XCBM
Value: $A054
Module: MPDOSK3
Format: XCBM <message>

Registers: In D1.L = Number
Out (A1) = String

Description: The CONVERT TO DECIMAL WITH MESSAGE primitive converts a 32-bit, signed number to a character string. The output string is preceded by the string whose PC relative address is in the operand field of the call.

The string can be up to 20 characters in length and is terminated by a null character. The number to be converted is passed to XCBM in data register D1. Address register A1 is returned with a pointer to the converted character string which is located in the monitor work buffer (MWB$) of the task control block.

Leading zeros are suppressed and the result ranges from -2147483648 to 2147483647.

The message address is a signed 16-bit PC relative address.

See also: None

Possible Errors: None

### 1.3.8 XCBP - CHECK FOR BREAK OR PAUSE

```
Mnemonic:       XCBP
Value:          $A074
Module:         MPDOSK2
Format:         XCBP
                &lt;status return&gt;

Registers:      Out  SR = EQ...No character
                          LT...[ESC]
                          LO...[CTRL-C]
                          NE...Pause
```

Note: If a 'BLT' instruction does not immediately follow the XCBP call, then the primitive exits to PDOS when an [ESC] character is entered.

If the ignore control character bit ($02) of the port flag is set, then XCBP always returns .EQ. status.

Description: The CHECK FOR BREAK OR PAUSE primitive looks for a character from your PRT$(A6) port. Any non-control character will cause XCBP to output a pause message and wait for another character.

The pause message consists of:

```
[CR]
     'Strike any key...'
[CR]
,
[CR].
```

A [CTRL-C] will abort any assigned console file and return the status 'LO'. If a 'BLT' instruction follows the XCBP primitive and an [ESC] character is entered, then the call returns with status 'LT'. Otherwise, an [ESC] will abort your program to VMEPROM.

An 'EQ' status indicates that no character was entered. An 'NE' status indicates a pause has occurred.

See also: None

Possible Errors: None

### 1.3.9 XCBX - CONVERT TO DECIMAL IN BUFFER

| | |
|---|---|
| Mnemonic: | XCBX |
| Value: | $A06A |
| Module: | MPDOSK3 |
| Format: | XCBX |

Registers: In D1.L = Number
(A1) = Buffer

Description: The CONVERT TO DECIMAL IN BUFFER primitive converts a 32-bit, 2’s complement number to a character string. The number to be converted is passed to XCBX in data register D1. Address register A1 points to the buffer where the converted string is stored.

Leading zeros are suppressed and a negative sign is the first character for negative numbers. The string is delimited by a null. The string has a maximum length of 11 characters and ranges from -2147483648 to 2147483647.

See also: 1.3.5 XCBD - CONVERT BINARY TO DECIMAL.

Possible Errors: None

### 1.3.10 XCDB - CONVERT ASCII TO BINARY

```
Mnemonic:      XCDB
Value:         $A056
Module:        MPDOSK3
Format:        XCDB
               <status return>

Registers:     In   (A1) = String
               Out  D0.B = Delimiter
                    D1.L = Number
                    (A1) = Updated string
                       SR = LT....No number
                            EQ....# w/o null delimiter
                            GT....#
```

Note: XCDB does not check for overflow.

Description: The CONVERT ASCII TO BINARY primitive converts an ASCII string of characters to a 32-bit, 2's complement number. The result is returned in data register D1 while the status register reflects the conversion results.

XCDB converts signed decimal, hexadecimal, or binary numbers.

Hexadecimal numbers are preceded by "$" and binary numbers by "%". A "-" indicates a negative number. There can be no embedded blanks.

An 'LT' status indicates that no conversion was possible. Data register D0 is returned with the first character and address register A1 points immediately after it.

A 'GT' status indicates that a conversion was made with a null delimiter encountered. The result is returned in data register D1. Address register A1 is returned with an updated pointer and register D0 is set to zero.

An 'EQ' status indicates that a conversion was made but the ASCII string was not terminated with a null character.

The result is returned in register D1 and the non-numeric, non-null character is returned in register D0.

Address register A2 has the address of the next character.

See also: None
Possible Errors: None

### 1.3.11 XCFA - CLOSE FILE W/ATTRIBUTE

```
Mnemonic:       XCFA
Value:          $A0D0
Module:         MPDOSF
Format:         XCFA
                <status error return>

Registers:      In   D1.W = File ID
                      D2.B = New attribute
```

Description: The CLOSE FILE WITH ATTRIBUTES primitive closes the open file specified by data register D1. At the same time, the file attributes are updated according to the byte contents of data register D2.

```
D2.B = $80   AC or Procedure file
     = $40   BN or Binary file
     = $20   OB or 68000 object file
     = $10   SY or 68000 memory image
     = $08   BX or BASIC binary token file
     = $04   EX or BASIC ASCII file
     = $02   TX or Text file
     = $01   DR or System I/O driver
     = $00   Clear file attributes
```

If the file was opened for sequential access and the file has been updated, then the END-OF-FILE marker is set at the current file pointer. If the file was opened for random or shared access, then the END-OF-FILE marker is updated only if the file has been extended (data was written after the current END-OF-FILE marker).

The LAST UPDATE is updated to the current date and time only if the file has been altered.

All files must be closed when opened! Otherwise, directory information and possibly even the file itself will be lost.

Note: If the file is not altered, then XCFA will not alter the file attributes.

See also:
1.3.72 XRFA - READ FILE ATTRIBUTES
1.3.109 XWFA - WRITE FILE ATTRIBUTES
1.3.110 XWFP - WRITE FILE PARAMETERS

Possible Errors:

52 = File not open
59 = Invalid file slot
75 = File locked
Disk errors

### 1.3.12 <u>XCHX - CONVERT BINARY TO HEX IN BUFFER</u>

| | |
|---|---|
| Mnemonic: | XCHX |
| Value: | $A068 |
| Module: | MPDOSK3 |
| Format: | XCHX |

Registers: In D1.L = Number
(A1) = Output buffer

Description: The CONVERT BINARY TO HEX IN BUFFER primitive converts a 32-bit number to its hexadecimal (base 16) representation. The number is passed in data register D1 and a pointer to a buffer in address register A1. The converted string consists of eight hexadecimal characters followed by a null.

See also: 1.3.6 XCBH - CONVERT BINARY TO HEX.

Possible Errors: None

### 1.3.13 XCLF - CLOSE FILE

| | |
|---|---|
| Mnemonic: | XCLF |
| Value: | $A0D2 |
| Module: | MPDOSF |
| Format: | XCLF<br><status error return> |

Registers: In D1.W = File ID

Description: The CLOSE FILE primitive closes the open file as specified by the file ID in data register D1. If the file was opened for sequential access and the file was updated, then the END-OF-FILE marker is set at the current file pointer.

If the file was opened for random or shared access, then the END-OF-FILE marker is updated only if the file was extended (ie. data was written after the current END-OF-FILE marker).

If the file has been altered, the current date and time is stored in the LAST UPDATE variable of the file directory. All open files must be closed at or before the completion of a task (or before disks are removed from the system)! Otherwise, directory information is lost and possibly even the file itself.

See also: None

Possible Errors:

52 = File not open
59 = Invalid slot #
75 = File locked
Disk errors

### 1.3.14 XCLS - CLEAR SCREEN

Mnemonic: XCLS
Value: $A076
Module: MPDOSK2
Format: XCLS

Registers: None

Note: The clear screen characters are located in the user TCB variable CSC$(A6).

Description: The CLEAR SCREEN primitive clears the console screen, homes the cursor, and clears the column counter. This function is adapted to the type of console terminals used in the PDOS system.

The character sequence to clear the screen is located in the task control block variable CSC$(A6). These characters are transferred from the parent task to the spawned task during creation. The initial characters come from the BIOS module.

If CSC$ is nonzero, then the CLEAR SCREEN primitive outputs up to four characters: one or two characters; an [ESC] followed by a character; or an [ESC], character, [ESC], and a final character. The one-word format allows for two characters. The parity bits cause the [ESC] character to precede each character.

If CSC$ is zero, then PDOS makes a call into the BIOS for custom clear screens. The entry point is B_CLS beyond the BIOS table.

The ST command maintains the CSC$ field, although it can be altered under program control.

See also: 1.3.67 XRCP - READ PORT CURSOR POSITION

Possible Errors: None

### 1.3.15 **XCPY - COPY FILE**

```
Mnemonic:      XCPY
Value:         $A0AE
Module:        MPDOSF
Format:        XCPY
               <status error return>

Registers:     In   (A1) = Source file name
                    (A2) = Destination file name
```

Note: A [CTRL-C] terminates this primitive and returns the error -1 in register D0.

Description: The COPY FILE primitive copies the source file into the destination file. The source file is pointed to by address register A1 and the destination file is pointed to by register A2. A [CTRL-C] halts the copy, prints '^C' to the console, and returns with error -1.

The file attributes of the source file are automatically transferred to the destination file.

See also: None

Possible Errors:

- -1 = Break file transfer
- 50 = Invalid file name
- 53 = File not defined
- 60 = File space full
- 61 = File already open
- 68 = Not PDOS disk
- 69 = No more file slots
- 70 = Position error
- Disk errors

### 1.3.16 XCTB - CREATE TASK BLOCK

```
Mnemonic:      XCTB
Value:         $A026
Module:        MPDOSK1
Format:        XCTB
               <status error return>

Registers:     In   D0.W = Task size (1 Kbyte increments)
                    D1.W = Task time.B/priority.B
                    D2.W = I/O port
                    (A0) = Optional low memory pointer
                    (A1) = Optional high memory pointer
                    (A2) = Command line pointer or entry address
               Out  D0.L = Spawned task number
```

Note: If D0.W is positive, A0 and A1 are undefined.

If D0.W equals zero, A0 and A1 are the new task's memory bounds and A2 contains the task's entry address.

If D0.W is negative, A0 and A1 are the new task's memory bounds and A2 points to the task's command line.

Description: The CREATE TASK primitive places a new task entry in the PDOS task list. Memory for the new task is either from the parent task or the system memory bit map. Data register D0 controls the creation mode of the new task as well as the task size. If register D0.W is positive, the first available contiguous memory block equal to D0.W (in 1 Kbyte) is allocated to the new task. If the block is not big enough, the upper memory of the parent task is allocated to the new task. The parent task's memory is then reduced by D0.W x 1 Kbytes. Address register A2 points to the new task command line. If A2 is zero, VMEPROM is invoked. If register D0.W is zero, registers A0 and A1 specify the new task's memory limits. Register A2 specifies the task's starting PC. The task control block begins at (A0) and is immediately followed by an XEXT primitive. The task user stack pointer is set at (A1). Thus, the new program should allow $1000 bytes at the low end and enough user stack space at the upper end.

If data register D0.W is negative, registers A0 and A1 specify the new task's memory limits. Register A2 points to the new task command line. (If A2=0, VMEPROM is invoked). The command line is transferred to the spawned program by a system message buffer. The maximum command line length is 64 characters. When the task is scheduled for the first time, message buffers are searched for a command. Messages with a source task equal to $FF are considered commands and moved to the task's monitor buffer.

The task CLI then processes the line. If no command message is found, then the VMEPROM is called directly.

Data register D1.W specifies the new task’s priority. The range is from 1 to 255. The larger the number, the higher the priority.

Data register D2.W specifies the I/O port to be used by the new task.

If register D2.W is positive, then the port is available for both input and output. If register D2.W is negative, then the port is used only for output. If register D2.W is zero, then no port is assigned. Only one task may be assigned to any one input port while many tasks may be assigned to an output port. Hence, a port is allocated for input only if it is available. An invalid port assignment does not result in an error.

A call is made to D$INT in the debugger module. This initializes all addresses, registers, breaks, and offsets.

Finally, the spawned task’s number is returned in register D0.L to the parent task. This can be used later to test task status or to kill the task.

See also: None

Possible Errors:

72 = Too many tasks
73 = Not enough memory

### 1.3.17 XDEV - DELAY SET/RESET EVENT

```
Mnemonic:      XDEV
Value:         $A032
Module:        MPDOSK1
Format:        XDEV
               <status error return>

Registers:     In   D0.L = Time
                    D1.B = Event (+=Set, -=Reset)

Note:          If D0.L=0, then the D1.B event is cleared.
```

Description: The DELAY SET/RESET EVENT primitive places a timed event in a system stack controlled by the system clock. Data register D0.L specifies the time interval in clock tics. When it counts to zero, then the event D1.B is set if positive, or reset if negative.

If the event already exists in the stack, it is replaced by the new entry. If the time specified in D0 equals zero, then any pending timed event equal to D1.B is deleted from the stack.

If D1.B is positive, event D1.B is first cleared. If D1.B is negative, event D1.B is set before exiting the primitive.

See also: 1.3.88 XSEF - SET EVENT FLAG W/SWAP
1.3.89 XSEV - SET EVENT FLAG
1.3.95 XSUI - SUSPEND UNTIL INTERRUPT
1.3.100 XTEF - TEST EVENT FLAG

Possible Errors:

83 = Delay event stack full

### 1.3.18 XDFL - DEFINE FILE

```
Mnemonic:    XDFL
Value:       $A0D4
Module:      MPDOSF
Format:      XDFL
             <status error return>

Registers: In   D0.W = # of contiguous sectors
                (A1) = File name
```

Description: The DEFINE FILE primitive creates a new file entry in a PDOS disk directory, specified by address register A1. A PDOS file name consists of an alphabetic character followed by up to 7 additional characters. An optional 3 character extension can be added if preceded by a colon. Likewise, the directory level and disk number are optionally specified by a semicolon and slash respectively. The file name is terminated with a null.

Data register D0 contains the number of sectors to be initially allocated at file definition. If register D0 is nonzero, then a contiguous file is created with D0 sectors. Otherwise, only one sector is allocated. Each sector of allocation corresponds to 252 bytes of data.

A contiguous file facilitates random access to file data since PDOS can directly position to any byte within the file without having to follow sector links. A contiguous file is automatically changed to a non-contiguous file if it is extended with non-contiguous sectors.

See also: None

Possible Errors:

50 = Invalid file name
51 = File already defined
55 = Fragmentation error
57 = File directory full
61 = File already open
68 = Not PDOS disk
Disk errors

### 1.3.19 <u>XDLF - DELETE FILE</u>

```
Mnemonic:    XDLF
Value:       $A0D6
Module:      MPDOSF
Format:      XDLF
             <status error return>

Registers: In   (A1) = File name
```

Description: The DELETE FILE primitive removes the file whose name is pointed to by address register A1 from the disk directory and releases all sectors associated with that file for use by other files on that same disk. A file cannot be deleted if it is delete (*) or write (**) protected.

See also: None

Possible Errors:

```
50 = Invalid file name
53 = File not defined
58 = File delete or write protected
61 = File already open
68 = Not PDOS disk
Disk errors
```

### 1.3.20 XDMP - DUMP MEMORY FROM STACK

```
Mnemonic:    XDMP
Value:       $A04A
Module:      MPDOSK3
Format:      XDMP

Registers: In   USP.L = <# of bytes>.W
                        <start address>.L
           Out  USP.L = USP.L + 6
```

Description: The DUMP MEMORY FROM STACK primitive dumps a block of memory to the console as specified by two parameters on the user stack (USP). The left side of the output is a hexadecimal dump and the right side is a masked ($7F) ASCII dump.

To use this primitive, first push a 32-bit address and then a 16-bit number of the amount of memory to be dumped. The primitive will automatically clean up the user stack.

See also: None

Possible Errors: None

### 1.3.21 XDPE - DELAY PHYSICAL EVENT

```
Mnemonic:    XDPE
Value:       $A114
Module:      MPDOSK1
Format:      XDPE

Registers: In   A0    = Event address
                D0.L  = Time TICs for delay (0=clear entry)
                D1.W  = Event descriptor
```

Description: XDPE causes the specified event to be set/cleared after the specified time has elapsed. Each event can have only one delayed action pending. Successive calls will supersede pending requests. Only the lower eight bits of the descriptor are used. To cancel pending actions, specify a delay time of 0.

The event descriptor is a 16-bit word that defines both the bit number at the specified A0 address and the action to take on the bit. The following bits are defined:

```
Bit number -- 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
              T  x  x  x  x  x x x S x x x x B B B
```

T = Should the bit be toggled on scheduling?
1 = Yes (toggle), 0 = No (do not toggle)

S = Suspend on event bit clear or set
1 = Suspend on SET, 0 = Suspend on CLEAR

BBB = The 680 x 0 bit number to use as an event
x = Reserved, should be 0

Since the bit number is specified in the lower three bits of the descriptor, you may use the descriptor with the 680 x 0 BTST, BCLR, BSET instructions.

See also: XDEV - Delay Set/Clear Event
XSOE - Suspend on Physical Event
XTLP - Translate Logical to Physical Event

## 1.3.22 XDTV - DEFINE TRAP VECTORS

```
Mnemonic:     XDTV
Value:        $A024
Module:       MPDOSK1
Format:       XDTV

Registers: In   D1.L = TVCZ FEDC BA98 7654 3210
                (A0) = Table base address
                (A1) = Vector table address

Vector table:   DC.L TRAP #0-<BASE ADR>
                ....
                DC.L TRAP #15-<BASE ADR>
                DC.L ZDIV-<BASE ADR>
                DC.L CHK-<BASE ADR>
                DC.L TRAPV-<BASE ADR>
                DC.L TRACE-<BASE ADR>

Note: The vector table size  is  variable  and  each
      entry corresponds to non-zero bits in the mask
      register (D1.L).  Each entry is a long  signed
      displacement from the base address register.

        D1.L = TVCZ FEDCBA9876543210
                \\\\ \              \_
                 \\\\ \________________ TRAPs #0-#15
                  \\\\_________________ Zero divide
                   \\\_________________ CHK
                    \\_________________ TRAPV
                     \_________________ Trace exception
```

Description:

The DEFINE TRAP VECTORS primitive loads user routine addresses into the task control block exception vector variables. Each task has the option to process its own TRAP, zero divide, CHK, TRAPV, and/or trace exceptions.

Data register D1 selects which vectors are to be loaded according to individual bits corresponding to vectors in the vector table pointed to by address register A1. Bits 0 through 19 (right to left) correspond to TRAPs 0 through 15, zero divide, CHK, TRAPV, and trace exceptions. A 1 bit moves a vector from the vector table (biased by base address A0) into the task control block.

When an exception occurs, the task control block is checked for a corresponding non-zero exception vector. If found, then the return address is pushed on the user stack (USP) followed by the exception address and condition codes. PDOS next moves to user mode and executes a return with condition codes (RTR). This effectively acts like a jump subroutine with the return address on the user stack.

The trace processing is handled differently. If the processor is in supervisor mode when a trace exception occurs, the trace bit is cleared and the exception is dismissed. The processor remains in supervisor mode. If the processor is in user mode and there is a non-zero trace variable in the task control block, then the trace is again disabled, the trace processor address is pushed on the supervisor stack along with status, and a return from exception is executed (RTE).

See also:

Possible Errors: None

## 1.3.23 XERR - RETURN ERROR D0 TO VMEPROM

Mnemonic: XERR
Value: $A00C
Module: MPDOSK1
Format: XERR

Registers: In D0.W = Error code

Description: The RETURN ERROR D0 TO VMEPROM primitive exits to VMEPROM and passes an error code in data register D0. PDOS prints 'PDOS ERR', followed by the decimal error number. The error call can be intercepted by changing the value of the ERR$ variable in the task TCB. This allows you to customize your own monitor.

See also: 1.3.24 XEXT - EXIT TO VMEPROM

Possible Errors: None

### 1.3.24 XEXC - EXECUTE PDOS CALL D7.W

Mnemonic: XEXC
Value: $A030
Module: MPDOSK1
Format: XEXC

Registers: In D7.W = Aline PDOS CALL

Description: The EXECUTE PDOS CALL D7.W primitive executes a variable PDOS primitive contained in data register D7. Any registers or error conditions apply to the corresponding PDOS call.

See also:

Possible Errors: Call dependent

### 1.3.25 XEXT - EXIT TO VMEPROM

Mnemonic: XEXT
Value: $A00E
Module: MPDOSK1
Format: XEXT
(Always exits to VMEPROM)

Registers: None

Description: The EXIT TO VMEPROM primitive exits a user program and returns to VMEPROM.

The exit can be intercepted by changing the value of the EXT$ variable in the task TCB. This primitive allows you to customize your own monitor.

See also: 1.3.22 XERR - RETURN ERROR D0 TO VMEPROM

Possible Errors: None

### 1.3.26 XFAC - FILE ALTERED CHECK

```
Mnemonic:    XFAC
Value:       $A0CE
Module:      MPDOSF
Format:      XFAC
             <status error return>

Registers: In   (A1) = FILE NAME
           Out    CC = File not altered
                  CS = File altered
                  NE = Error
```

Description: The FILE ALTERED CHECK primitive looks at the altered bit (bit $80) of the file pointed to by address register A1. If the bit is zero (not altered), then the primitive returns with the carry status bit clear.

If the alter bit is set (file altered), then it is cleared and the primitive returns with carry set. If either case, the bit is always cleared.

See also: None

Possible Errors: Disk errors

### 1.3.27 **XFBF - FLUSH BUFFERS**

Mnemonic: XFBF
Value: $A0F8
Module: MPDOSF
Format: XFBF
<status error return>

Registers: None

Description: The FLUSH BUFFERS primitive forces all file slots with active channel buffers to write any updated data to the disk. It thus does a checkpoint of any open and altered file.

See also: None

Possible Errors: Disk errors

### 1.3.28 XFFN - FIX FILE NAME

```
Mnemonic:    XFFN
Value:       $A0A0
Module:      MPDOSF
Format:      XFFN
             <status error return>

Registers: In   (A1) = File name
           Out  D0.L = Disks(4th/3rd/2nd/1st)
                (A1) = MWB$, Fixed file name
```

Description: The FIX FILE NAME primitive parses a character string for file name, extension, directory level, and disk number. The results are returned in the 32-character monitor work buffer (MWB$(A6)). Data register D0 is also returned with the disk number. The error return is used for an invalid file name.

The monitor work buffer is cleared and the following assignments are made:

```
     0(A1) = File name
     8(A1) = File extension
    11(A1) = File directory level
```

System defaults are used for the disk number and file directory level when they are not specified in the file name.

See also: 1.3.70 XRDN - READ DIRECTORY ENTRY BY NAME

Possible Errors:

50 = Invalid file name

### 1.3.29 XFTD - FIX TIME & DATE

Mnemonic: XFTD
Value: $A058
Module: MPDOSK3
Format: XFTD

Registers: Out D0.W = Hours * 256 + Minutes
D1.W = (Year * 16 + Month) * 32 + Day

Description: The FIX TIME & DATE primitive returns a two-word encoded time and date generated from the system timers. The resultant codes include month, day, year, hours, and minutes. The ordinal codes can be sorted and used as inputs to the UNPACK DATE (XUDT) and UNPACK TIME (XUTM) primitives.

Data register D0.W contains the time and register D1.W contains the date. This format is used throughout PDOS for time stamping items.

See also:
1.3.52 XPAD - PACK ASCII DATE
1.3.71 XRDT - READ DATE
1.3.84 XRTM - READ TIME
1.3.101 XUAD - UNPACK ASCII DATE
1.3.102 XUDT - UNPACK DATE
1.3.106 XUTM - UNPACK TIME

Possible Errors: None

### 1.3.30 XFUM - FREE USER MEMORY

Mnemonic: XFUM
Value: $A040
Module: MPDOSK1
Format: XFUM
<status error return>

Registers: In D0.W = Number of K bytes
(A0) = Beginning address

Description: The FREE USER MEMORY primitive deallocates user memory to the system memory bit map. Data register D0.W specifies how much memory is to be deallocated while address register A0 points to the beginning of the data block.

Memory thus deallocated is available for any task use including new task creation.

Possible Errors:

79 = Memory error

### 1.3.31 XGCB - CONDITIONAL GET CHARACTER

```
Mnemonic:    XGCB
Value:       $A048
Module:      MPDOSK2
Format:      XGCB
             <status return>

Registers: Out  D0.L = Character in bits 0-7
                  SR = EQ....No character
                       LO....[CTRL-C]
                       LT....[ESC]
                       MI....[CTRL-C] or [ESC]
```

Note: If the ignore control character bit ($02) of the port flag is set, then XGCB ignores [CTRL-C] and [ESC].

Description: The CONDITIONAL GET CHARACTER primitive checks for a character from first, the input message pointer (IMP$(A6)), second, the assigned input file (ACI$(A6)), and then finally, the interrupt driven input character buffer (PRT$(A6)). If a character is found, it is returned in the right byte of data register D0.L and the rest of the register is cleared.

If there is no input message, no assigned console port character, and the interrupt buffer is empty, the status is returned as 'EQ'.

The status is returned 'LO' and the break flag cleared if the returned character is a [CTRL-C]. The input buffer is also cleared. Thus, all characters entered after the [CTRL-C] and before the XGCB call are dropped.

The status is returned 'LT' and the break flag cleared if the returned character is the [ESC] character.

For all other characters, the status is returned 'HI' and 'GT'. The break flag is not affected.

Possible Errors: None

### 1.3.32 XGCC - GET CHARACTER CONDITIONAL

```
Mnemonic:    XGCC
Value:       $A078
Module:      MPDOSK2
Format:      XGCC
             <status return>

Registers: Out  D0.L = Character in bits 0-7
                  SR = EQ....No character
                       LO....[CTRL-C]
                       LT....[ESC]
                       MI....[CTRL-C] or [ESC]
```

Note: If the ignore control character bit ($02) of the port flag is set, then XGCC ignores [CTRL-C] and [ESC].

Description: The GET CHARACTER CONDITIONAL primitive checks the interrupt driven input character buffer and returns the next character in the right byte of data register D0.L. The rest of the register is cleared. The input buffer is selected by the input port variable (PRT$) of the TCB.

If the buffer is empty, the 'EQ' status bit is set. If the character is a [CTRL-C], then the break flag and input buffer are cleared, and the status is returned 'LO'. If the character is the [ESC] character, then the break flag is cleared and the status is returned 'LT'.

If no special character is encountered, the character is returned in register D0 and the status set 'HI' and 'GT'.

If no port has been assigned for input (ie. port 0 or phantom port), then the routine always returns an 'EQ' status.

Possible Errors: None

### 1.3.33 XGCP - GET PORT CHARACTER

```
Mnemonic:    XGCP
Value:       $A09E
Module:      MPDOSK2
Format:      XGCP
             <status return>

Registers: Out  D0.L = Character in bits 0-7
                  SR = LO....[CTRL-C]
                       LT....[ESC]
                       MI....[CTRL-C] or [ESC]
```

Note: If the ignore control character bit ($02) of the port flag is set, then XGCP ignores [CTRL-C] and [ESC].

Description: The GET PORT CHARACTER primitive checks for a character in the interrupt driven input character buffer. If a character is found, it is returned in the right byte of data register D0.L and the rest of the register is cleared. The input buffer is selected by the input port variable (PRT$) of the TCB.

If the interrupt buffer is empty, the task is suspended pending a character interrupt.

The status is returned 'LO' and the break flag cleared if the returned character is a [CTRL-C]. The input buffer is also cleared. Thus, all characters entered after the [CTRL-C] and before the XGCR call are dropped.

The status is returned 'LT' and the break flag cleared if the returned character is the [ESC] character.

For all other characters, the status is returned 'HI' and 'GT'. The break flag is not affected.

If no port has been assigned for input, (ie. port 0 or phantom port), then an error 86 occurs.

Possible Errors: None

## 1.3.34 XGCR - GET CHARACTER

```
Mnemonic:    XGCR
Value:       $A07A
Module:      MPDOSK2
Format:      XGCR
             <status return>

Registers: Out  D0.L = Character in bits 0-7
                  SR = LO....[CTRL-C]
                       LT....[ESC]
                       MI....[CTRL-C] or [ESC]
```

Note: If the ignore control character bit ($02) of the port flag is set, then XGCR ignores [CTRL-C] and [ESC].

Description: The GET CHARACTER primitive checks for a character from first, the input message pointer (IMP$(A6)); second, the assigned input file (ACI$(A6)); and then finally, the interrupt driven input character buffer (PRT$(A6)). If a character is found, it is returned in the right byte of data register D0.L and the rest of the register is cleared.

If there is no input message, no assigned console port character, and the interrupt buffer is empty, the task is suspended pending a character interrupt.

The status is returned 'LO' and the break flag cleared if the returned character is a [CTRL-C]. The input buffer is also cleared. Thus, all characters entered after the [CTRL-C] and before the XGCR call are dropped.

The status is returned 'LT' and the break flag cleared if the returned character is the [ESC] character.

For all other characters, the status is returned 'HI' and 'GT'. The break flag is not affected.

If no port has been assigned for input, (ie. port 0 or phantom port), then an error 86 occurs.

Possible Errors: None

### 1.3.35 XGLB - GET LINE IN BUFFER

```
Mnemonic:    XGLB
Value:       $A07C
Module:      MPDOSK2
Format:      XGLB
             {BLT.x ESCAPE}   optional
             <status return>

Registers: In   (A1) = Buffer address
           Out  D1.L = Number of characters
                  SR = EQ...[CR] only
                       LT...[ESC]
                       LO...[CTRL-C]
```

Note: If the ignore control character bit ($02) of the port flag is set, then XGLB ignores [CTRL-C] and [ESC].

Description: The GET LINE IN BUFFER primitive gets a character line into the buffer pointed to by address register A1. The XGCR primitive is used by XGLB and hence characters can come from a memory message, a file, or the task console port.

The buffer must be at least 80 characters in length. The line is delimited by a carriage return. The status returns EQUAL if only a [CR] is entered.

If an [ESC] is entered, the task exits to VMEPROM unless a 'BLT' instruction immediately follows the XGLB call. If such is the case, then XGLB returns with status set at 'LT'.

If the assigned console flag (ACI$(A6)) is set, then the '&' character is used for character substitutions. '&0' is replaced with the last system error number. '&1' is replaced with the first parameter of the command line, '&2' with the second, and so forth up to '&9'.

The command line can be edited with various system defined control characters. A [BACKSPACE] ($08) moves the cursor one character to the left. A [CTRL-F] ($0C) moves the cursor one character to the right. A [RUB] ($7F) deletes one character to the left. A [CTRL-D] ($04) deletes the character under the cursor. The cursor need not be at the end of the line when the [CR] is entered.

See also: 1.3.36 XGLU - GET LINE IN USER BUFFER

Possible Errors: None

### 1.3.36 XGLM - GET LINE IN MONITOR BUFFER

```
Mnemonic:    XGLM
Value:       $A07E
Module:      MPDOSK2
Format:      XGLM
             {BLT.x ESCAPE}   optional
             <status return>

Registers: Out  (A1) = String
                D1.L = Number of characters
                  SR = EQ...[CR] only
                       LT...[ESC]
                       LO...[CTRL-C]
```

Note: If the ignore control character bit ($02) of the port flag is set, then XGLM ignores [CTRL-C] and [ESC].

Description:

The GET LINE IN MONITOR BUFFER primitive gets a character line into the monitor buffer located in the task control block. The XGCR primitive is used by XGLM and hence, characters can come from a memory message, a file, or the task console port.

The buffer has a maximum length of 80 characters and is delimited by a carriage return. The status returns EQUAL if only a [CR] is entered. If an [ESC] is entered, the task exits to VMEPROM unless a 'BLT' instruction immediately follows the XGLM call. If such is the case, then XGLM returns with status set at 'LT'.

If the assigned console flag (ACI$(A6)) is set, then the '&' character is used for character substitutions. '&0' is replaced with the last system error number. '&1' is replaced with the first parameter of the command line, '&2' with the second, and so forth up to '&9'.

The command line can be edited with various system-defined control characters. A [BACKSPACE] ($08) moves the cursor one character to the left. A [CTRL-L] ($0C) moves the cursor one character to the right. A [RUB] ($7F) deletes one character to the left. A [CTRL-D] ($04) deletes the character under the cursor. The cursor need not be at the end of the line when the [CR] is entered.

The last command line can be recalled to the buffer by entering a [CTRL-A] ($01). This line can then be edited using the above control characters.

Possible Errors: None

### 1.3.37 XGLU - GET LINE IN USER BUFFER

```
Mnemonic:    XGLU
Value:       $A080
Module:      MPDOSK2
Format:      XGLU
             {BLT.x ESCAPE   ;optional}
             <status return>

Registers: Out  (A1) = String
                D1.L = Number of characters
                  SR = EQ...[CR] only
                       LT...[ESC]
                       LO...[CTRL-C]
```

Note: If the ignore control character bit ($02) of the port flag is set, then XGLU ignores [CTRL-C] and [ESC].

Description:

The GET LINE IN USER BUFFER primitive gets a character line into the user buffer. Address register A6 normally points to the user buffer. The XGCR primitive is used by XGLU; hence, characters come from a memory message, a file, or the task console port. The line is delimited by a carriage return. The status returns EQUAL if only a [CR] is entered. Address register A1 is returned with a pointer to the first character.

The user buffer is located at the beginning of the task control block and is 256 characters in length. However, the XGLU routine limits the number of input characters to 78 plus two nulls.

If an [ESC] ($1B) is entered, the task exits to VMEPROM unless a 'BLT' instruction immediately follows the XGLU call. If such is the case, then XGLU returns with status set at 'LT'.

If the assigned console flag (ACI$(A6)) is set, then the '&' character is used for character substitutions. '&0' is replaced with the last system error number. '&1' is replaced with the first parameter of the command line, '&2' with the second, and so forth up to '&9'.

The command line can be edited with various system defined control characters. A [BACKSPACE] ($08) moves the cursor one character to the left. A [CTRL-L] ($0C) moves the cursor one character to the right. A [RUB] ($7F) deletes one character to the left. A [CTRL-D] ($04) deletes the character under the cursor. The cursor need not be at the end of the line when the [CR] is entered.

Possible Errors: None

### 1.3.38 XGML - GET MEMORY LIMITS

Mnemonic: XGML
Value: $A010
Module: MPDOSK1
Format: XGML

Registers: Out (A0) = End TCB (TBE$)
(A1) = Upper memory limit (EUM$-USZ)
(A2) = Last loaded address (BUM$)
(A5) = System RAM (SYRAM)
(A6) = Task TCB

Description: The GET MEMORY LIMITS subroutine returns the user task memory limits. These limits are defined as the first usable location after the task control block ($500 beyond address register A6) and the end of the user task memory. The task may use up to but not including the upper memory limit.
Address register A0 is returned pointing to the beginning of user storage (which is the end of the TCB). Register A1 points to the upper task memory limit less $100 hexadecimal bytes for the user stack pointer (USP). Register A2 is the last loaded memory address as provided by the PDOS loader. Address registers A5 and A6 are returned with the pointers to system RAM (SYRAM) and the task control block (TCB).

Possible Errors: None

### 1.3.39 <u>XGMP - GET MESSAGE POINTER</u>

```
Mnemonic:    XGMP
Value:       $A004
Module:      MPDOSK1
Format:      XGMP
             <status return>

Registers: In D0.L = Message slot number (0..15)
          Out D0.L = Source task # (-1 = no message)
                SR = EQ....Message (Event[64+Message slot#]=0)
                     NE....No message
              D0.L = Error number 83 if no message
              (A1) = Message
```

Description: The GET MESSAGE POINTER primitive looks for a task message pointer. If no message is ready, then data register D0 returns with a minus one (-1) and status is set to 'Not Equal'.

If a message is waiting, then data register D0 returns with the source task number, address register A1 returns with the message pointer, event (64 + message slot #) is set to zero indicating message received, and status is returned equal.

See also:

1.3.40 XGTM - GET TASK MESSAGE
1.3.44 XKTM - KILL TASK MESSAGE
1.3.90 XSMP - SEND MESSAGE POINTER
1.3.93 XSTM - SEND TASK MESSAGE

Possible Errors:

83 = Message slot empty

## 1.3.40 XGNP - GET NEXT PARAMETER

```
Mnemonic:     XGNP
Value:        $A05A
Module:       Emulated by VMEPROM
Format:       XGNP
              <status return>

Registers: Out  SR = LO....No parameter
                           [(A1)=0]
                     EQ....Null Parameter
                           [(A1)=0]
                     HI....Parameter
                           [(A1)=PARAMETER]
```

Description: The GET NEXT PARAMETER primitive parses the VMEPROM command buffer for the next command parameter. The XGNP primitive clears all leading spaces of a parameter. A parameter is a character string delimited by a space, comma, period, or null. If a parameter begins with a left parenthesis, then all parsing stops until a matching right parenthesis or null is found. Hence, spaces, commas, and periods are passed in a parameter when enclosed in parentheses. Parentheses may be nested to any depth.

A 'LO' status is returned if the last parameter delimiter is a null or period. XGNP does not parse past a period. In this case, address register A1 is returned pointing to a null string.

An 'EQ' status is returned if the last parameter delimiter is a comma and no parameter follows. Address register A1 is returned pointing to a null string.

A 'HI' status is returned if a valid parameter is found. Address register A1 then points to the parameter.

Possible Errors: None

### 1.3.41 XGTM - GET TASK MESSAGE

```
Mnemonic:    XGTM
Value:       $A01E
Module:      MPDOSK1
Format:      XGTM
             <status return>

Registers: In   (A1) = Buffer address
           Out  D0.L = Source task #
                        (-1 = no message)
                  SR = EQ....message found
                       NE....no message
```

Description: The GET TASK MESSAGE primitive searches the PDOS message buffers for a message with a destination equal to the current task number. If a message is found, it is moved to the buffer pointed to by address register A1. The message buffer is then released, and the status is set EQUAL. If no message is found, status is returned NE.

The buffer must be at least 64 bytes in length. (This is a configuration parameter.) The message buffers are serviced on a first in, first out basis (FIFO). Messages are data independent and pass any type of binary data.

See also:
1.3.38 XGMP - GET MESSAGE POINTER
1.3.44 XKTM - KILL TASK MESSAGE
1.3.90 XSMP - SEND MESSAGE POINTER
1.3.93 XSTM - SEND TASK MESSAGE

Possible Errors: None

## 1.3.42 XGUM - GET USER MEMORY

```
Mnemonic:    XGUM
Value:       $A03E
Module:      MPDOSK1
Format:      XGUM
             <status error return>

Registers: In   D0.W = Number of K bytes
           Out  (A0) = Beginning memory address
                (A1) = End memory address
```

Description: The GET USER MEMORY primitive searches the system memory bit map for a contiguous block of memory equal to D0.W Kbytes. If found, the 'EQ' status is set, address registers A0 and A1 are returned the start and end memory address, and the memory block is marked as allocated in the bit map.

See also: 1.3.29 XFUM - FREE USER MEMORY

Possible Errors:

73 = Not enough memory

### 1.3.43 XISE - INITIALIZE SECTOR

```
Mnemonic:    XISE
Value:       $A0C0
Module:      MPDOSF
Format:      XISE
             <status error return>

Registers: In   D0.B = Disk number
                D1.W = Logical sector number
                (A2) = Buffer address
```

Description: The INIT SECTOR primitive is a system-defined, hardware-dependent program which writes 256 bytes of data from a buffer (A2) to a logical sector number (D1) on disk (D0). This routine is meant to be used only for disk initialization and is equivalent to the WRITE SECTOR (XWSE) primitive for all sectors except 0. Sector 0 is not checked for the PDOS ID code.

See also:

1.3.79 XRSE - READ SECTOR
1.3.82 XRSZ - READ SECTOR ZERO
1.3.112 XWSE - WRITE SECTOR

Possible Errors:

Disk errors

## 1.3.44 XKTB - KILL TASK

Mnemonic: XKTB
Value: $A0FA
Module: MPDOSK1
Format: XKTB
<status error return>

Registers: In D0.B = Task number

Note: If D0.B equals zero, then kill current task. If D0.B is negative, then kill task without allocating task memory to system bit map.

Description: The KILL TASK primitive removes a task from the PDOS task list and optionally returns the task's memory to the system memory bit map. Only the current task or a task spawned by the current task can be killed. Task 0 cannot be killed.

The kill process includes releasing the input port assigned to the task and closing all files associated with the task.

The task number is specified in data register D0.B. If register D0.B equals zero, then the current task is killed and its memory deallocated in the system memory bit map.

If D0.B is positive, then the selected task is killed and its memory deallocated. If D0.B is negative, then task number ABS(D0.B) is killed, but its memory is not deallocated in the memory bit map.

See also: 1.3.16 XCTB - CREATE TASK BLOCK

Possible Errors:

74 = No such task
76 = Task locked

### 1.3.45 <u>XKTM - KILL TASK MESSAGE</u>

```
Mnemonic:    XKTM
Value:       $A028
Module:      MPDOSK1
Format:      XKTM
             <status return>

Registers: In   D0.B = Task #
                (A1) = Buffer address
           Out  D0.L = Source task #
                        (-1 = no message)
                  SR = EQ....message found
                       NE....no message
```

Description: The KILL TASK MESSAGE primitive allows you to read (and thus clear) any task’s messages from the system message buffers.

See also:

1.3.38 XGMP - GET MESSAGE POINTER
1.3.40 XGTM - GET TASK MESSAGE
1.3.90 XSMP - SEND MESSAGE POINTER
1.3.93 XSTM - SEND TASK MESSAGE

Possible Errors: None

## 1.3.46 XLDF - LOAD FILE

```
Mnemonic:    XLDF
Value:       $A0B0
Module:      MPDOSF
Format:      XLDF
             <status error return>

Registers: In   D1.B = Execution flag
                (A0) = Start of load memory
                (A1) = End of load memory
                (A3) = File name
            Out (A0) = EAD$ - Lowest loaded address
                (A1) = BUM$ - Last loaded address

Note:     If D1.B=0, then XLDF returns to your calling program.
          If  D1.B<>0, then the program is immediately executed.
```

Description: The LOAD FILE primitive reads and loads 68000 object code into user memory. The file name pointer is passed in address register A3. Registers A0 and A1 specify the memory bounds for the relocatable load. The file must be typed 'OB' or 'SY'. If data register D1.B is zero, then XLDF returns to the calling program. Otherwise, the loaded program is immediately executed.

The 68000 object should be position-independent section 0 code without any external references or definitions.

A 'SY' file is generated from an 'OB' file by the MSYFL utility. The condensed object is a direct memory image and must be position-independent code.

The XLDF primitive uses long word moves and may move up to three bytes more than contained in an 'SY' file. As such, you must allow for extra space for data moves to an existing program.

Possible Errors:

63 = Illegal object tag
64 = Illegal section
65 = File not loadable
71 = Exceeds task size
73 = Not enough memory
Disk errors

### 1.3.47 XLER - LOAD ERROR REGISTER

Mnemonic: XLER
Value: $A03A
Module: MPDOSK1
Format: XLER

Registers: In D0.W = Error number

Description: The LOAD ERROR REGISTER primitive stores data register D0.W in the task control block variable LEN$(A6). This variable will replace the parameter substitution variable '&0' during a procedure file.

User programs should execute this call when an error occurs.

The enable echo flag (ECF$(A6)) is cleared by this call.

Possible Errors: None

### 1.3.48 XLFN - LOOK FOR NAME IN FILE SLOTS

```
Mnemonic:    XLFN
Value:       $A0A2
Module:      MPDOSF
Format:      XLFN
             <status return>

Registers: In   D0.B = Disk number
                (A1) = Fixed file name
           Out  D3.W = File ID (Disk #/Index)
                (A3) = Slot entry address
                  SR = NE...File name not found
                       EQ...File name found
```

Note: If D3.W=0, then no slots are available.

Description: The LOOK FOR NAME IN FILE SLOTS primitive searches through the file slot table for the file name as specified by registers D0.B and A1. If the name is not found, register D3.W returns with a -1 or 0. The latter indicates the file was not found and there are no more slots available. Otherwise, register D3.W returns the associated file ID and register A3 returns the address of the file slot.

A file slot is a 38-byte buffer where the status of an open file is maintained. There are 32 file slots available. The file ID consists of the disk # and the file slot index.

File slots assigned to read-only files are skipped and not considered for file match.

Possible Errors: None

### 1.3.49 XLKF - LOCK FILE

Mnemonic: XLKF
Value: $A0D8
Module: MPDOSF
Format: XLKF
<status error return>

Registers: In D1.W = File ID

Description: The LOCK FILE primitive locks an opened file so that no other task can gain access until an UNLOCK FILE (XULF) primitive is executed. Only the locking task has access to the locked file.

A locked file is indicated by a -1 ($FF) in the left byte of the lock file parameter (LF) of the file slot usage (FS) command. The locking task number is stored in the left byte of the task number parameter (TN).

See also: 1.3.103 XULF - UNLOCK FILE

Possible Errors:

52 = File not open
59 = Invalid slot #
75 = File locked
Disk errors

### 1.3.50 <u>XLKT - LOCK TASK</u>

```
Mnemonic:    XLKT
Value:       $A014
Module:      MPDOSK1
Format:      XLKT
             <status return>

Registers: Out  SR = EQ...Not locked
                     NE...Locked
```

Description: The LOCK TASK primitive locks the requesting task in the run state by setting the swap lock variable in system RAM to nonzero. The task remains locked until an UNLOCK TASK (XULT) is executed. The status of the lock variable BEFORE the call is returned in the status register.

XLKT waits until all locks (Level 2 and Level 3 locks) are cleared before the task is locked.

See also: 1.3.104 XULT - UNLOCK TASK

Possible Errors: None

### 1.3.51 **XLSR - LOAD STATUS REGISTER**

Mnemonic: XLSR
Value: $A02E
Module: MPDOSK1
Format: XLSR

Registers: In D1.W = 68000 status register

Description: The LOAD STATUS REGISTER primitive allows you to directly load the 68000 status register. Of course, only appropriate bits (i.e. the interrupt mask too high, supervisor mode, trace mode, etc.) are to be set so that the system is not crashed.

See also: 1.3.96 XSUP - ENTER SUPERVISOR MODE

Possible Errors: None

### 1.3.52 XNOP - OPEN SHARED RANDOM FILE

```
Mnemonic:    XNOP
Value:       $A0DA
Module:      MPDOSF
Format:      XNOP
             <status error return>

Registers: In   (A1) = File name
           Out  D0.W = File attribute
                D1.W = File ID
```

Notes: Uses multiple directory file search. You MUST lock and position file before each multi-task access.

Description: The OPEN SHARED RANDOM FILE primitive opens a file for shared random access by assigning the file to an area of system memory called a file slot. The file ID and file attribute are returned to the calling program in registers D1 and D0, respectively. Thereafter, the file is referenced by the file ID and not by the file name. A new entry in the file slot table is made only if the file is not already opened for shared access.

The file ID (returned in register D1) is a 2-byte number. The left byte is the disk number and the right byte is the file slot index. The file attributes are returned in register D0.

The END-OF-FILE marker on a shared file is changed only when the file has been extended. All data transfers are buffered through a channel buffer; data movement to and from the disk is by full sectors.

An "opened count" is incremented each time the file is shared-opened and is decremented by each close operation. The file is only closed by PDOS when the count is zero. This count is saved in the right byte of the locked file parameter (LF) and is listed by the file slot usage command (FS).

Possible Errors:

50 = Invalid file name
53 = File not defined
60 = File space full
61 = File already open
68 = Not PDOS disk
69 = Not enough file slots
Disk errors

### 1.3.53 <u>XPAD - PACK ASCII DATE</u>

```
Mnemonic:        XPAD
   Value:        $A00A
  Module:        MPDOSK3
  Format:        XPAD

Registers:  In   (A1) = 'DY-MON-YR'
            Out  D1.W = (Year*16+month)*32+day
                         (YYYY YYYM MMMD DDDD)
                 (A1) = Updated
                   SR = .EQ. - Conversion ok
                        .NE. - Error
```

Description: The PACK ASCII DATE primitive converts an ASCII date string to an encoded binary number in data register D1. The result is compatible with other PDOS date primitives such as XUAD.

See Also:

1.3.28 XFTD - FIX TIME & DATE
1.3.71 XRDT - READ DATE
1.3.84 XRTM - READ TIME
1.3.101 XUAD - UNPACK ASCII DATE
1.3.102 XUDT - UNPACK DATE

Possible Errors: Status errors.

## 1.3.54 XPBC - PUT BUFFER TO CONSOLE

Mnemonic: XPBC
Value: $A084
Module: MPDOSK2
Format: XPBC

Registers: None

Description: The PUT USER BUFFER TO CONSOLE primitive outputs the ASCII contents of the user buffer to the user console and/or SPOOL file. The output string is delimited by the null character. The user buffer is the first 256 bytes of the task control block and is pointed to by address register A6. With the exception of control characters and characters with the parity bit on, each character increments the column counter by one. A [BACKSPACE] ($08) decrements the counter while a [CR] ($0D) clears the counter. [TAB]s ($09) are expanded with blanks to MOD 8 character zone fields. If there are coinciding bits in the unit (UNT$(A6)) and spool unit (SPU$(A6)) variables of the TCB, then the processed characters are written to the spool unit file slot (SPI$(A6)) and are not sent to the corresponding output ports. If a disk error occurs in the spool file, then all subsequent output characters echo as a bell until the error is corrected by selecting a different UNIT or resetting the SPOOL UNIT.

See also: 1.3.34 XGLB - GET LINE IN BUFFER

Possible Errors: None

### 1.3.55 XPCC - PUT CHARACTER(S) TO CONSOLE

Mnemonic: XPCC
Value: $A086
Module: MPDOSK2
Format: XPCC

Registers: In D0.W = Character(s)

Description: The PUT CHARACTER TO CONSOLE primitive outputs one or two ASCII characters in data register D0 to the user console and/or SPOOL file. The right byte (bits 0 through 7) is first and is followed by the left byte (bits 8 through 15) if non-zero. If the right byte or both bytes are zero, nothing is output to the console.

With the exception of control characters and characters with the parity bit on, each character increments the column counter by one. A [BACKSPACE] ($08) decrements the counter while a [CR] ($0D) clears the counter. [TAB]s ($09) are expanded with blanks to MOD 8 character zone fields.

If there are coinciding bits in the unit (UNT$(A6)) and spool unit (SPU$(A6)) variables of the TCB, then the processed characters are written to the spool unit file slot (SPI$(A6)) and are not sent to the corresponding output ports. If a disk error occurs in the spool file, then all subsequent output characters echo as a bell until the error is corrected by selecting a different UNIT or resetting the SPOOL UNIT.

See also:
1.3.56 XPCR - PUT CHARACTER RAW
1.3.57 XPDC - PUT DATA TO CONSOLE

Possible Errors: None

### 1.3.56 <u>XPCL - PUT CRLF TO CONSOLE</u>

Mnemonic: XPCL
Value: $A088
Module: MPDOSK2
Format: XPCL

Registers: None

Description: The PUT CRLF TO CONSOLE primitive outputs the ASCII characters carriage return <$0A> and line feed <$0D> to the user console and/or SPOOL file. The column counter is cleared.

If there are coinciding bits in the unit (UNT$(A6)) and spool unit (SPU$(A6)) variables of the TCB, then the processed characters are written to the spool unit file slot (SPI$(A6)) and are not sent to the corresponding output ports. If a disk error occurs in the spool file, then all subsequent output characters echo as a bell until the error is corrected by selecting a different UNIT or resetting the SPOOL UNIT.

Possible Errors: None

### 1.3.57 XPCP - PLACE CHARACTER IN PORT BUFFER

```
Mnemonic:    XPCP
Value:       $AOBC
Module:      MPDOSK2
Format:      XPCP

Registers: In   D0.B  = Character to insert
                D1.W  = Input port number (1 to 15)
           Out  SR    = .EQ.  = High water (character is inserted)
                        .NE.  = Character is inserted
```

Description: XPCP allows a character to be placed into the input buffer of any VMEPROM port from a task or program.

**Note:** Once the status returns EQ (high water)_, subsequent XPCP calls will return a status of NE as if everything were normal, but the data is discarded. Once the status of EQ is detected, the transmitting task should monitor the status of the port with the XRPS (read port status) call until bit 56 is cleared.

The port specified in the XPCP call is independent of window g - it refers to the physical port, not the logical port.

### 1.3.58 XPCR - PUT CHARACTER RAW

Mnemonic: XPCR
Value: $A0BA
Module: MPDOSK2
Format: XPCR

Registers: In D0.B = CHARACTER

Description: The PUT CHARACTER RAW primitive outputs the character in the lower byte of data register D0 to the user console. No attempt is made by PDOS to interpret control characters.

See also:
1.3.54 XPCC - PUT CHARACTER(S) TO CONSOLE
1.3.57 XPDC - PUT DATA TO CONSOLE

Possible Errors: None

### 1.3.59 XPDC - PUT DATA TO CONSOLE

Mnemonic: XPDC
Value: $A096
Module: MPDOSK2
Format: XPDC

Registers: In D7.W = LENGTH
(A1) = DATA STRING

Description: The PUT DATA TO CONSOLE primitive outputs data-independent bytes to the console. Address register A1 points to the string while data register D7 has the string length.

If there are coinciding bits in the unit (UNT$(A6)) and spool unit (SPU$(A6)) variables of the TCB, then the processed characters are written to the spool unit file slot (SPI$(A6)) and are not sent to the corresponding output ports. If a disk error occurs in the spool file, then all subsequent output characters echo as a bell until the error is corrected by selecting a different UNIT or resetting the SPOOL UNIT.

See also:
1.3.54 XPCC - PUT CHARACTER(S) TO CONSOLE
1.3.56 XPCR - PUT CHARACTER RAW

Possible Errors: None

### 1.3.60 XPEL - PUT ENCODED LINE TO CONSOLE

Mnemonic: XPEL
Value: $A06E
Module: MPDOSK2
Format: XPEL

Registers: In (A1) = Message

Description: The PUT ENCODED LINE TO CONSOLE primitive outputs to the user console the message pointed to by address register A1. An encoded message is similar to any other string with the exception that the parity bit is used to output blanks and the character $80 outputs a carriage return/line feed.

If the parity bit is set and the masked character ($7F) is less than or equal to a blank, then the numeric value of the negated character is used as the number of blanks to be inserted in the output stream. If the mask character is greater than a blank, then that character is output followed by one blank.

With the exception of control characters, each character increments the column counter by one. A [BACKSPACE] ($08) decrements the counter while a [CR] ($0D) clears the counter. [TAB]s ($09) are expanded with blanks to MOD 8 character zone fields.

If there are coinciding bits in the unit (UNT$(A6)) and spool unit (SPU$(A6)) variables of the TCB, then the processed characters are written to the spool unit file slot (SPI$(A6)) and are not sent to the corresponding output ports. If a disk error occurs in the spool file, then all subsequent output characters echo as a bell until the error is corrected by selecting a different UNIT or resetting the SPOOL UNIT.

See also:
- 1.3.59 XPEM - PUT ENCODED MESSAGE TO CONSOLE
- 1.3.60 XPLC - PUT LINE TO CONSOLE
- 1.3.61 XPMC - PUT MESSAGE TO CONSOLE

Possible Errors: None

### 1.3.61 XPEM - PUT ENCODED MESSAGE TO CONSOLE

Mnemonic: XPEM
Value: $A09C
Module: MPDOSK2
Format: XPEM <message>

Registers: None

Description: The PUT ENCODED MESSAGE TO CONSOLE primitive outputs to the user console the PC relative message contained in the word following the call. An encoded message is similar to any other string with the exception that the parity bit is used to output blanks and the character $80 outputs a carriage return/line feed.

If the parity bit is set and the masked character ($7F) is less than or equal to a blank, then the numeric value of the negated character is used as the number of blanks to be inserted in the output stream. If the mask character is greater than a blank, then that character is output followed by one blank.

With the exception of control characters, each character increments the column counter by one. A [BACKSPACE] ($08) decrements the counter while a [CR] ($0D) clears the counter. [TAB]s ($09) are expanded with blanks to MOD 8 character zone fields.

If there are coinciding bits in the unit (UNT$(A6)) and spool unit (SPU$(A6)) variables of the TCB, then the processed characters are written to the spool unit file slot (SPI$(A6)) and are not sent to the corresponding output ports. If a disk error occurs in the spool file, then all subsequent output characters echo as a bell until the error is corrected by selecting a different UNIT or resetting the SPOOL UNIT.

See also:
- 1.3.58 XPEL - PUT ENCODED LINE TO CONSOLE
- 1.3.60 XPLC - PUT LINE TO CONSOLE
- 1.3.61 XPMC - PUT MESSAGE TO CONSOLE

Possible Errors: None

### 1.3.62 XPLC - PUT LINE TO CONSOLE

Mnemonic: XPLC
Value: $A08A
Module: MPDOSK2
Format: XPLC

Registers: In (A1) = ASCII string

Description: The PUT LINE TO CONSOLE primitive outputs the ASCII character string pointed to by address register A1 to the user console and/or SPOOL file. The string is delimited by the null character.

With the exception of control characters and characters with the parity bit on, each character increments the column counter by one. A [BACKSPACE] ($08) decrements the counter while a [CR] ($0D) clears the counter. [TAB]s ($09) are expanded with blanks to MOD 8 character zone fields.

If there are coinciding bits in the unit (UNT$(A6)) and spool unit (SPU$(A6)) variables of the TCB, then the processed characters are written to the spool unit file slot (SPI$(A6)) and are not sent to the corresponding output ports. If a disk error occurs in the spool file, then all subsequent output characters echo as a bell until the error is corrected by selecting a different UNIT or resetting the SPOOL UNIT.

See also:
1.3.58 XPEL - PUT ENCODED LINE TO CONSOLE
1.3.59 XPEM - PUT ENCODED MESSAGE TO CONSOLE
1.3.61 XPMC - PUT MESSAGE TO CONSOLE

Possible Errors: None

### 1.3.63 XPMC - PUT MESSAGE TO CONSOLE

Mnemonic: XPMC
Value: $A08C
Module: MPDOSK2
Format: XPMC <message>

Registers: None

Description: The PUT MESSAGE TO CONSOLE primitive outputs the ASCII character string pointed to by the message address word immediately following the PDOS call to the user console and/or SPOOL file. The address is a PC relative 16-bit displacement to the message. The output string is delimited by the null character.

With the exception of control characters and characters with the parity bit on, each character increments the column counter by one. A [BACKSPACE] ($08) decrements the counter while a [CR] ($0D) clears the counter. [TAB]s ($09) are expanded with blanks to MOD 8 character zone fields.

If there are coinciding bits in the unit (UNT$(A6)) and spool unit (SPU$(A6)) variables of the TCB, then the processed characters are written to the spool unit file slot (SPI$(A6)) and are not sent to the corresponding output ports. If a disk error occurs in the spool file, then all subsequent output characters echo as a bell until the error is corrected by selecting a different UNIT or resetting the SPOOL UNIT.

See also:
- 1.3.58 XPEL - PUT ENCODED LINE TO CONSOLE
- 1.3.59 XPEM - PUT ENCODED MESSAGE TO CONSOLE
- 1.3.60 XPLC - PUT LINE TO CONSOLE

Possible Errors: None

### 1.3.64 XPSC - POSITION CURSOR

Mnemonic: XPSC
Value: $A08E
Module: MPDOSK2
Format: XPSC

Registers: In D1.B = Row
D2.B = Column

Note: Uses PSC$(A6) as lead characters.

Description: The POSITION CURSOR primitive positions the cursor on the console terminal according to the row and column values in data registers D1 and D2. Register D1 specifies the row on the terminal and generally ranges from 0 to 23, with 0 being the top row. Register D2 specifies the column of the terminal and ranges from 0 to 79, with 0 being the left-hand column. Register D2 is also loaded into the column counter reflecting the true column of the cursor.

The XPSC primitive outputs either one or two leading characters followed by the row and column. The leading characters output by XPSC are located in PSC$(A6) of the task control block. These characters are transferred from the parent task to the spawned task during creation. The initial characters come from the BIOS module.

The row and column characters are biased by $20 if the parity bit of the first character is set. Likewise, if the second character's parity bit is set, then row/column order is reversed. This accommodates most terminal requirements for positioning the cursor.

If PSC$ is zero, then PDOS makes a call into the BIOS for custom position cursor. The entry point is B_PSC beyond the BIOS table.

The ST command of the user interface can be used to change the position cursor codes.

See also:
1.3.14 XCLS - CLEAR SCREEN
1.3.67 XRCP - READ PORT CURSOR POSITION

Possible Errors: None

### 1.3.65 XPSF - POSITION FILE

Mnemonic: XPSF
Value: $A0DC
Module: MPDOSF
Format: XPSF
<status error return>

Registers: In D1.W = File ID
D2.L = Byte position

Note: A byte position equal to -1 positions to the end of the file.

Description: The POSITION FILE primitive moves the file byte pointer to any byte position within a file. The file ID is given in register D1 and the long word byte position is specified in register D2.

An error occurs if the byte position is greater than the current end-of-file marker.

A contiguous file greatly enhances the speed of the position primitive since the desired sector is directly computed. However, the position primitive does work with non-contiguous files, as PDOS follows the sector links to the desired byte position.

A contiguous file is extended by positioning to the end-of-file marker and writing data. However, PDOS will alter the file type to non-contiguous if a contiguous sector is not available. This would result in random access being much slower.

See also:
1.3.73 XRFP - READ FILE POSITION
1.3.87 XRWF - REWIND FILE

Possible Errors:

52 = File not open
59 = Invalid slot #
70 = Position error
Disk errors

### 1.3.66 <u>XPSP - PUT SPACE TO CONSOLE</u>

Mnemonic: XPSP
Value: $A098
Module: MPDOSK2
Format: XPSP

Registers: None

Description: The PUT SPACE TO CONSOLE outputs a [SP] ($20) character to the user console. There are no registers or status involved. If there are coinciding bits in the unit (UNT$(A6)) and spool unit (SPU$(A6)) variables of the TCB, then the processed characters are written to the spool unit file slot (SPI$(A6)) and are not sent to the corresponding output ports. If a disk error occurs in the spool file, then all subsequent output characters echo as a bell until the error is corrected by selecting a different UNIT or resetting the SPOOL UNIT.

See also: 1.3.54 XPCC - PUT CHARACTER(S) TO CONSOLE

Possible Errors: None

### 1.3.67 XRBF - READ BYTES FROM FILE

```
Mnemonic:    XRBF
Value:       $A0DE
Module:      MPDOSF
Format:      XRBF
             <status error return>

Registers: In   D0.L = Number of bytes
                D1.W = File ID
                (A2) = R/W buffer address
           Out  D3.L = Number of bytes read
                       (On EOF only.)
```

Description: The READ BYTES FROM FILE primitive reads the number of bytes specified in register D0 from the file specified by the file ID in register D1 into a memory buffer pointed to by address register A2. If the channel buffer has been rolled to disk, the least-used buffer is freed and the desired buffer is restored to memory. The file slot ID is placed on the top of the last-access queue.

If an error occurs during the read operation, the error return is taken with the error number in register D0 and the number of bytes actually read in register D3.

The read is independent of the data content. The buffer pointer in register A2 is on any byte boundary. The buffer is not terminated with a null.

A byte count of zero in register D0 results in one byte being read from the file. This facilitates single byte data acquisition.

See also:


1.3.74 XRLF - READ LINE FROM FILE
1.3.107 XWBF - WRITE BYTES TO FILE
1.3.111 XWLF - WRITE LINE TO FILE


Possible Errors:

52 = File not open
56 = End of file
59 = Invalid slot #
Disk errors

### 1.3.68 **XRCN - RESET CONSOLE INPUTS**

Mnemonic: XRCN
Value: $A0B2
Module: MPDOSF
Format: XRCN

Registers: None

Description: The RESET CONSOLE INPUTS closes the current procedure file. If there are other procedure files pending (nested), then they become active again.

See also: 1.3.4 XCBC - CHECK FOR BREAK CHARACTER

Possible Errors: None

### 1.3.69 XRCP - READ PORT CURSOR POSITION

```
Mnemonic:    XRCP
Value:       $A092
Module:      MPDOSK2
Format:      XRCP

Registers: In   D0.W = Port #
           Out  D1.L = Row
                D2.L = Column
```

Note: If D0.W=0, then the current port (PRT$(A6)) is used.

Description: The READ PORT CURSOR POSITION primitive reads the current cursor position for the port designated by data register D0.B. The PDOS system maintains a column count (0-79) and a row count (0-23) for each port. When the cursor reaches row 23, the count is not incremented, acting like a screen scroll.

See also:

1.3.14 XCLS - CLEAR SCREEN
1.3.62 XPSC - POSITION CURSOR

Possible Errors: None

### 1.3.70 XRDE - READ NEXT DIRECTORY ENTRY

```
Mnemonic:     XRDE
Value:        $A0A6
Module:       MPDOSF
Format:       XRDE
              <status error return>

Registers: In   D0.B = Disk number
                D1.B = Read flag (0=1st)
                (A2) = Last 32 byte directory entry
                TW1$ = Sector number
                TW2$ = number of directory entries
           Out  D1.W = Sector number
                (A2) = Next entry
```

Description: The READ NEXT DIRECTORY ENTRY primitive reads sequentially through a disk directory. If register D1.B is zero, then the routine begins with the first directory entry. If register D1.B is nonzero, then based on the last directory entry (pointed to by register A2), the next entry is read.

The calling routine must maintain registers D0.B and A2, the user I/O buffer, and temporary variables TW1$ and TW2$ of the task control block between calls to XRDE.

Possible Errors:

53 = File not defined (End of directory)
68 = Not PDOS disk
Disk errors

### 1.3.71 XRDM - DUMP REGISTERS

Mnemonic: XRDM
Value: $A02A
Module: MPDOSK1
Format: XRDM

Registers: In All

Description: The DUMP REGISTERS primitive formats and outputs all the current register values of the 68000 to the user console along with the program counter, status register, and the supervisor stack.

The registers and status are not affected by this primitive.

See also: 1.3.20 XDMP - DUMP MEMORY FROM STACK

Possible Errors: None

### 1.3.72 XRDN - READ DIRECTORY ENTRY BY NAME

```
Mnemonic:    XRDN
Value:       $A0A8
Module:      MPDOSF
Format:      XRDN
             <status error return>

Registers: In   D0.B = Disk number
                MWB$ = File name
           Out  D1.W = Sector number in memory
                (A2) = Directory entry
                TW2$ = Entry count
```

Description: The READ DIRECTORY ENTRY BY NAME primitive reads directory entries by file name. Register D0.B specifies the disk number. The file name is located in the Monitor Work Buffer (MWB$) in a fixed format. Several other parameters are returned in the monitor TEMP storage of the user task control block. These variables assist in the housekeeping operations on the disk directory.

See also: 1.3.27 XFFN - FIX FILE NAME

Possible Errors:

53 = File not defined
68 = Not PDOS disk
Disk errors

### 1.3.73 XRDT - READ DATE

Mnemonic: XRDT
Value: $A05C
Module: MPDOSK3
Format: XRDT

Registers: Out (A1) = 'MN/DY/YR'<null>

Description: The READ DATE primitive returns the current system date as a nine character string. The format is 'MN/DY/YR' followed by a null. Address register A1 points to the string in the monitor work buffer.

See also:
- 1.3.28 XFTD - FIX TIME & DATE
- 1.3.52 XPAD - PACK ASCII DATE
- 1.3.84 XRTM - READ TIME
- 1.3.101 XUAD - UNPACK ASCII DATE
- 1.3.102 XUDT - UNPACK DATE
- 1.3.106 XUTM - UNPACK TIME

Possible Errors: None

### 1.3.74 XRFA - READ FILE ATTRIBUTES

```
Mnemonic:     XRFA
Value:        $A0E0
Module:       MPDOSF
Format:       XRFA
              <status error return>

Registers: In   (A1) = File name
           Out  (A2) = Directory entry
                D0.L = Disk number
                D1.L = File size (in bytes)
                D2.L = Level/attributes
```

Note: Uses multiple directory file search.

Description: The READ FILE ATTRIBUTES primitive returns the disk number of where the file was found in data register D0.L. Data register D1.L is returned with the size of the file in bytes. The file directory level is returned in the upper word of register D2.L and the file attributes are returned in register D2.W. The file name is pointed to by address register A1. File attributes are defined as follows:

```
$80xx   AC - Procedure file
$40xx   BN - Binary file
$20xx   OB - 68000 object file
$10xx   SY - 68000 memory image
$08xx   BX - BASIC binary token file
$04xx   EX - BASIC ASCII file
$02xx   TX - Text file
$01xx   DR - System I/O driver
$xx04   C  - Contiguous file
$xx02   *  - Delete protect
$xx01   ** - Delete and write protect
```

See also:

1.3.11 XCFA - CLOSE FILE W/ATTRIBUTE
1.3.109 XWFA - WRITE FILE ATTRIBUTES
1.3.110 XWFP - WRITE FILE PARAMETERS

Possible Errors:

50 = Invalid file name
53 = File not defined
60 = File space full
Disk errors

### 1.3.75 XRFP - READ FILE POSITION

```
Mnemonic:    XRFP
Value:       $A0FE
Module:      MPDOSF
Format:      XRFP
             <status error return>

Registers: In   D1.W = File ID
           Out  (A3) = File slot address
                D2.L = Byte position
                D3.L = EOF byte position
```

Description: The READ FILE POSITION primitive returns the current file position, end-of-file position, and file slot address. The open file is selected by the file ID in data register D1.W.

Address register A3 is returned pointing to the open file slot. Data registers D2.L and D3.L are returned with the current file byte position and the end-of-file position respectively.

See also:

1.3.63 XPSF - POSITION FILE
1.3.87 XRWF - REWIND FILE

Possible Errors:

52 = File not open
59 = Invalid slot #
Disk errors

### 1.3.76 XRLF - READ LINE FROM FILE

```
Mnemonic:    XRLF
Value:       $A0E2
Module:      MPDOSF
Format:      XRLF
             <status error return>

Registers: In   D1.W = File ID
                (A2) = R/W buffer address
           Out  D3.L = # of bytes read
                       (On EOF only.)
```

Description: The READ LINE primitive reads one line, delimited by a carriage return [CR], from the file specified by the file ID in register D1. If a [CR] is not encountered after 132 characters, then the line and primitive are terminated. Address register A2 points to the buffer in user memory where the line is to be stored. If the channel buffer has been rolled to disk, the least-used buffer is freed and the buffer is restored to memory. The file slot ID is placed on the top of the last-access queue.

If an error occurs during the read operation, the error return is taken with the error number in register D0 and the number of bytes actually read in register D3.

The line read is dependent upon the data content. All line feeds ([LF]) are dropped from the data stream and the [CR] is replaced with a null. The buffer pointer in register A2 may be on any byte boundary. The buffer is not terminated with a null on an error return.

See also:

1.3.65 XRBF - READ BYTES FROM FILE
1.3.107 XWBF - WRITE BYTES TO FILE
1.3.111 XWLF - WRITE LINE TO FILE

Possible Errors:

52 = File not open
56 = End of file
59 = Invalid slot #
Disk errors

### 1.3.77 XRNF - RENAME FILE

```
Mnemonic:     XRNF
Value:        $A0E4
Module:       MPDOSF
Format:       XRNF
              <status error return>

Registers: In   (A1) = Old file name
                (A2) = New file name
```

Description: The RENAME FILE primitive renames a file in a PDOS disk directory. The old file name is pointed to by address register A1. The new file name is pointed to by address register A2.

The XRNF primitive is used to change the directory level for any file by letting the new file name be a numeric string equivalent to the new directory level. XRNF first attempts a conversion on the second parameter before renaming the file. If the string converts to a number without error, then only the level of the file is changed.

See also:

1.3.18 XDFL - DEFINE FILE
1.3.19 XDLF - DELETE FILE

Possible Errors:

50 = Invalid file name
51 = File already defined
Disk errors

### 1.3.78 **XROO - OPEN RANDOM READ ONLY FILE**

```
Mnemonic:    XROO
Value:       $A0E6
Module:      MPDOSF
Format:      XROO
             <status error return>

Registers: In   (A1) = File name
           Out  D0.W = File attribute
                D1.W = File ID
```

Note: Uses multiple directory file search.

Description: The OPEN RANDOM READ ONLY FILE primitive opens a file for random access by assigning the file to an area of system memory called a file slot, and returning a file ID and file attribute to the calling program. Thereafter, the file is referenced by the file ID and not by the file name. This type of file open provides read only access.

The file ID (returned in register R1) is a 2-byte number. The left byte is the disk number and the right byte is the channel buffer index. The file attribute is returned in register D0.

Since the file cannot be altered, it cannot be extended nor is the LAST UPDATE parameter changed when it is closed. All data transfers are buffered through a channel buffer and data movement to and from the disk is by full sectors.

A new file slot is allocated for each XROO call even if the file is already open. The file slot is allocated beginning with slot 1 to 32.

Possible Errors:

50 = Invalid file name
53 = File not defined
61 = File already open
68 = Not PDOS disk
69 = Not enough file slots
Disk errors

## 1.3.79 XROP - OPEN RANDOM

```
Mnemonic:    XROP
Value:       $A0E8
Module:      MPDOSF
Format:      XROP
             <status error return>

Registers: In   (A1) = File name
           Out  D0.W = File attribute
                D1.W = File ID
```

Note: Uses multiple directory file search.

Description: The OPEN RANDOM FILE primitive opens a file for random access by assigning the file to an area of system memory called a file slot, and returning a file ID and file attribute to the calling program. Thereafter, the file is referenced by the file ID and not by the file name.

The file ID (returned in register D1) is a 2-byte number. The left byte is the disk number and the right byte is the channel buffer index. The file attribute is returned in register D0.

The END-OF-FILE marker on a random file is changed only when the file has been extended. All data transfers are buffered through a channel buffer and data movement to and from the disk is by full sectors.

The file slot is allocated beginning with slot 32 to slot 1. If the file is already open, then the file slot is shared.

Possible Errors:

```
50 = Invalid file name
53 = File not defined
61 = File already open
68 = Not PDOS disk
69 = Not enough file slots
Disk errors
```

### 1.3.80 <u>XRPS - READ PORT STATUS</u>

```
Mnemonic:    XRPS
Value:       $A094
Module:      MPDOSK2
Format:      XRPS
             <status error return>

Registers: In   D0.W = Port number
           Out  D1.L = ACI$.W / portflag.B / Status.B
```

Note: If D0.W=0, then the current port (PRT$(A6)) is used.

Description: The READ PORT STATUS primitive reads the current status of the port specified by data register D0.W. The high order word of data register D1.L is returned zero if no procedure file is open. Otherwise, it is returned with ACI$.

The low order word is returned with the port flag bits and the status as returned for the port UART routine. The flag bits indicate if eight bit I/O is occurring, if DTR or ^S ^Q protocol is in effect, and other flags.

See also:

1.3.3 XBCP - BAUD CONSOLE PORT
1.3.92 XSPF - SET PORT FLAG

Possible Errors:

66 = Invalid port or baud rate

### 1.3.81 XRSE - READ SECTOR

```
Mnemonic:    XRSE
Value:       $A0C2
Module:      MPDOSF
Format:      XRSE
             <status error return>

Registers: In   D0.B = Disk number
                D1.W = Sector number
                (A2) = Buffer pointer
```

Description: The READ SECTOR primitive calls a system-defined, hardware-dependent program which reads 256 bytes of data into a memory buffer pointed to by address register A2. The disk is selected by data register D0. Register D1 specifies the logical sector number to be read.

See also:

1.3.42 XISE - INITIALIZE SECTOR
1.3.82 XRSZ - READ SECTOR ZERO
1.3.112 XWSE - WRITE SECTOR

Possible Errors:

Disk errors

### 1.3.82 XRSR - READ STATUS REGISTER

Mnemonic: XRSR
Value: $A042
Module: MPDOSK1
Format: XRSR

Registers: Out D0.W = 68000 status register

Description: The READ STATUS REGISTER primitive allows you to read the 68000 status register. Of course, this is equivalent to the 'MOVE.W SR,Dx' instruction on the 68000. However, this instruction is privileged on the 68010 and 68020. Hence, it is advisable to use the XRSR primitive to read the status register to make software upward compatible.

Possible Errors: None

### 1.3.83 XRST - RESET DISK

```
Mnemonic:    XRST
Value:       $A0B4
Module:      MPDOSF
Format:      XRST

Registers: In   D1.W = -1.... Reset by task
                       >=0... Reset by disk
```

Description: The RESET DISK primitive closes all open files either by task or disk number. The primitive also clears the assigned input file ID. If register D1 equals -1, then all files associated with the current task are closed. Otherwise, register D1 specifies a disk and all files opened on that disk are closed.

XRST has no error return and as such, closes all files even though errors occur in the close process. This is necessary to allow for recovery from previous errors.

See also:

1.3.11 XCFA - CLOSE FILE W/ATTRIBUTE
1.3.13 XCLF - CLOSE FILE

Possible Errors: None

### 1.3.84 XRSZ - READ SECTOR ZERO

```
Mnemonic:    XRSZ
Value:       $A0C4
Module:      MPDOSF
Format:      XRSZ
             <status error return>

Registers: In   D0.B = Disk number
           Out  D1.L = 0
                (A2) = User buffer pointer (A6)
```

Description: The READ SECTOR ZERO primitive is a system-defined, hardware-dependent program which reads 256 bytes of data into the user memory buffer (usually pointed to by address register A6). The disk is selected by data register D0.W. Register D1.L is cleared and logical sector zero is read.

See also:

1.3.42 XISE - INITIALIZE SECTOR
1.3.79 XRSE - READ SECTOR
1.3.112 XWSE - WRITE SECTOR

Possible Errors:

Disk errors

### 1.3.85 XRTE - RETURN FROM INTERRUPT

Mnemonic: XRTE
Value: $A044
Module: MPDOSK1
Format: XRTE

Registers: In SSP = Status register.W
Program counter.L

Description: The RETURN FROM INTERRUPT primitive is used to return from an interrupt process routine with a context switch. This allows an immediate rescheduling of the highest priority ready task which may be suspended pending the occurrence of an event set by the interrupt routine.

If the interrupted system is locked when the XRTE primitive is executed, then the reschedule flag (RFLG.(A5)) is cleared and a return from exception instruction (RTE) is executed. When the system clears the task lock, RFLG. is tested and set (TAS) and a rescheduling occurs at that time.

Possible Errors: None

### 1.3.86 **XRTM - READ TIME**

```
Mnemonic:    XRTM
Value:       $A05E
Module:      MPDOSK3
Format:      XRTM

Registers: Out      (A1) = 'HR:MN:SC'<null>
                10(A1).W = Tics/second (B.TPS)
                12(A1).L = Tics (TICS.)
```

Description: The READ TIME primitive returns the current time as a nine-character string. The format is 'HR:MN:SC' followed by a null. Address register A1 points to the string in the monitor work buffer.

See also:

1.3.28 XFTD - FIX TIME & DATE
1.3.52 XPAD - PACK ASCII DATE
1.3.71 XRDT - READ DATE
1.3.101 XUAD - UNPACK ASCII DATE
1.3.102 XUDT - UNPACK DATE
1.3.106 XUTM - UNPACK TIME

Possible Errors: None

### 1.3.87 XRTP - READ TIME PARAMETERS

```
Mnemonic:     XRTP
Value:        $A034
Module:       MPDOSK1
Format:       XRTP

Registers: Out  D0.L = TICS.
                D1.L = MONTH/DAY/YEAR/0
                D2.L = HOURS/MINUTES/SECONDS/0
                D3.L = B.TPS
```

Description: The READ TIME PARAMETERS primitive returns the current time parameters. Data register D0 returns with the current tic count (TICS.(A5)). Register D1.L returns with the current date and register D2.L the current time. Both are three bytes that are left-justified. Finally, data register D3.L returns with the number of clock tics per second.

See also:

1.3.28 XFTD - FIX TIME & DATE
1.3.52 XPAD - PACK ASCII DATE
1.3.71 XRDT - READ DATE
1.3.84 XRTM - READ TIME
1.3.101 XUAD - UNPACK ASCII DATE
1.3.102 XUDT - UNPACK DATE
1.3.101 XUTM - UNPACK TIME

Possible Errors: None

## 1.3.88 **XRTS - READ TASK STATUS**

```
Mnemonic:    XRTS
Value:       $A012
Module:      MPDOSK1
Format:      XRTS
             <status return>

Registers: In   D0.W = Task number
           Out  D1.L = 0 - Not executing
                     = +N - Time slice
                     = -N - (Event #1/Event #2)
                A0.L = TLST entry (IF -D0: A0=TLST.)
                  SR = Status of D1.L
```

Note: If D0.W=-1, then the current task number is returned in D1.L.

Description: The READ TASK STATUS primitive returns in register D1 and the status register returns the time parameter of the task specified by register D0. The time reflects the execution mode of the task. If D1 returns zero, then the task is not in the task list. If D1 returns a value greater than zero, then the task is in the run state (executing). If D1 returns a negative value, then the task is suspended pending event -(D1).

The task number is returned from the CREATE TASK BLOCK (XCTB) primitive. It can also be obtained by setting data register D0 equal to a minus one. In this case, register D1.L is returned with the current task number.

See also: 1.3.94 XSTP - SET/READ TASK PRIORITY

Possible Errors: None

### 1.3.89 <u>XRWF - REWIND FILE</u>

Mnemonic: XRWF
Value: $A0EA
Module: MPDOSF
Format: XRWF
<status error return>

Registers: In D1.W = File ID

Description: The REWIND FILE primitive positions the file specified by the file ID in register D1, to byte position zero.

See also:
1.3.63 XPSF - POSITION FILE
1.3.73 XRFP - READ FILE POSITION

Possible Errors:

52 = File not open
59 = Invalid slot #
70 = Position error
Disk errors

### 1.3.90 XSEF - SET EVENT FLAG W/SWAP

```
Mnemonic:    XSEF
Value:       $A018
Module:      MPDOSK1
Format:      XSEF
             <status return>

Registers: In   D1.B = Event (+=Set, -=Reset)
           Out    SR = NE....Set
                       EQ....Reset
```

Note: An XSWP is automatically executed after the event is set or reset. Event 128 is local to each task.

If D1.B is positive, then the event is set.
If D1.B is negative, then the event is reset.

Description: The SET EVENT FLAG WITH SWAP primitive sets or resets an event flag bit. The event number is specified in data register D1.B and is module 128. If the content of register D1.B is positive, then the event bit is set to 1. Otherwise, the bit is reset to 0. Event 128 can only be set. (It is cleared by the task scheduler.)

The status of the event bit prior to changing the event is returned in the status register. If the event was 0, then the 'EQ' status is returned. Also, an immediate context switch occurs thus scheduling any higher priority task pending on that event.

Events are summarized as follows:

```
   1-63 = Software events
  64-80 = Software resetting events
  81-95 = Output port events
 96-111 = Input port events
    112 = 1/5 second event
    113 = 1 second event
    114 = 10 second event
    115 = 20 second event
    116 = TTA active
    117 = LPT active
```

See also:

1.3.17 XDEV - DELAY SET/RESET EVENT
1.3.89 XSEV - SET EVENT FLAG
1.3.95 XSUI - SUSPEND UNTIL INTERRUPT
1.3.100 XTEF - TEST EVENT FLAG

Possible Errors: None

## 1.3.91 XSEV - SET EVENT FLAG

```
Mnemonic:    XSEV
Value:       $A046
Module:      MPDOSK1
Format:      XSEV
             <status return>

Registers: In   D1.B = Event (+=Set, -=Reset)
           Out    SR = NE....Set
                       EQ....Reset
```

Note: Event 128 is local to each task.

If D1.B is positive, then the event is set.
If D1.B is negative, then the event is reset.

Description: The SET EVENT FLAG primitive sets or resets an event flag bit. The event number is specified in data register D1.B and is module 128. If the content of register D1.B is positive, then the event bit is set to 1. Otherwise, the bit is reset to 0. Event 128 can only be set. (It is cleared by the task scheduler.)

The status of the event bit prior to changing the event is returned in the status register. If the event was 0, then the 'EQ' status is returned. A context switch DOES NOT occur with this call making it useful for interrupt routines outside the PDOS system.

Events are summarized as follows:

```
    1-63 = Software events
   64-80 = Software resetting events
   81-95 = Output port events
  96-111 = Input port events
     112 = 1/5 second event
     113 = 1 second event
     114 = 10 second event
     115 = 20 second event
     116 = TTA active
     117 = LPT active
```

See also:


1.3.17 XDEV - DELAY SET/RESET EVENT
1.3.89 XSEV - SET EVENT FLAG
1.3.95 XSUI - SUSPEND UNTIL INTERRUPT
1.3.100 XTEF - TEST EVENT FLAG


Possible Errors: None

### 1.3.92 XSMP - SEND MESSAGE POINTER

```
Mnemonic:    XSMP
Value:       $A002
Module:      MPDOSK1
Format:      XSMP
             <status return>

Registers: In   D0.B = Message slot number (0..15)
                (A1) = Message
           Out    SR = EQ....Message sent (Event[64+slot #]=1)
                       NE....No message sent
```

Description: The SEND MESSAGE POINTER primitive sends a 32-bit message to the message slot specified by data register D0.B. Address register A1 contains the message. If there is still a message pending, then the primitive immediately returns with status set 'Not Equal' and D0.L equal to 83. Otherwise, the message is taken by PDOS event (64 + message slot number) is set to one indicating a message is ready, and status is returned 'Equal'.

The primitive XSMP is only valid for message slots 0 through 15. (This is because of current event limitations.)

See also:

1.3.38 XGMP - GET MESSAGE POINTER
1.3.40 XGTM - GET TASK MESSAGE
1.3.44 XKTM - KILL TASK MESSAGE
1.3.93 XSTM - SEND TASK MESSAGE

Possible Errors:

83 = Message buffer pending

### 1.3.93 XSOE - SUSPEND ON PHYSICAL EVENT

Mnemonic: XSOE
Value: $A112
Module: MPDOSK1
Format: XSOE

```
Registers: In   D1.L  = Event 1 Descriptor.w, Event 0 Descriptor.w
                A0    = Event 0 address (0=no event 0 to suspend on)
                A1    = Event 1 address (0=no event 1 to suspend on)
           Out  D0    = -1 if awaken on event 0;1 if awaken on event 1
```

Note: This call is the same as XSUI but with physical events.

Description: XSOE allows a task to suspend on one or two events within the system. Tasks that suspend on physical events are listed as suspended on events -1/1. If event 0 is the scheduling event, a -1 is returned; otherwise, a 1 is returned.

The event descriptor is a 16 bit word that defines both the bit number at the specified A0,A1 address and the action to take o n the bit. The following bits are defined:

```
Bit number -- 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
0
            T  x  x  x  x  x x x S x x x x B B B
```

T = Should the bit be toggled on scheduling?
1 = Yes (toggle), 0 = No (do not toggle)

S = Suspend on event bit clear or set
1 = Suspend on SET, 0 = Suspend on CLEAR

BBB = The 680 x 0 bit number to use as an event
x = Reserved, should be 0

Since the bit number is specified in the lower three bits of the descriptor, you may use the descriptor with the 680x0 BTST,BCLR,BSET instructions.

See also: XDPE - Delay On Physical Event
XTLP - Translate Logical To Physical Event

### 1.3.94 XSOP - OPEN SEQUENTIAL FILE

```
Mnemonic:     XSOP
Value:        $A0EC
Module:       MPDOSF
Format:       XSOP
              <status error return>

Registers: In    (A1) = File name
           Out  D0.W = File attribute
                D1.W = File ID
```

Note: Uses multiple directory file search.

Description: The OPEN SEQUENTIAL FILE primitive opens a file for sequential access by assigning the file to an area of system memory called a file slot and returning a file ID and file type to the calling program. Thereafter, the file is referenced by the file ID and not by the file name.

The file ID (returned in register D1) is a 2-byte number. The left byte is the disk number and the right byte is the file slot index. The file attribute is returned in D0.

The END-OF-FILE marker on a sequential file is changed whenever data is written to the file. All data transfers are buffered through a channel buffer; data movement to and from the disk is by full sectors.

The file slots are allocated beginning with slot 32 down to slot 1.

Possible Errors:

- 50 = Invalid file name
- 53 = File not defined
- 61 = File already open
- 68 = Not PDOS disk
- 69 = Not enough file slots
- Disk errors

### 1.3.95 XSPF - SET PORT FLAG

```
Mnemonic:    XSPF
Value:       $A09A
Module:      MPDOSK2
Format:      XSPF
             <status error return>

Registers: In   D0.W = Port number
                D1.B = Port flag (fwpi8dcs)
           Out  D1.B = Old port flag
```

Note: If D0.W=0, then the current port (PRT$(A6)) is used.

Description: The SET PORT FLAG primitive stores the port flag passed in data register D1.B in the port flag register as specified by register D0.W. If flag bits 'p', 'i', or '8' change, the BIOS baud port routine is called.

See also:
1.3.3 XBCP - BAUD CONSOLE PORT
1.3.78 XRPS - READ PORT STATUS

Possible Errors:

66 = Invalid port or baud rate

### 1.3.96 XSTM - SEND TASK MESSAGE

```
Mnemonic:    XSTM
Value:       $A020
Module:      MPDOSK1
Format:      XSTM
             <status error return>

Registers: In   D0.B = TASK NUMBER
                (A1) = MESSAGE
```

Description: The SEND TASK MESSAGE primitive places a 64-character message into a PDOS system message buffer. The message is data-independent and is pointed to by address register A1.

Data register D0 specifies the destination of the message. If register D0 is negative, and there is no input port (phantom port), then the message is sent to the parent task. If there is a port, then the message is sent to itself and will appear at the next command line. Otherwise, register D0 specifies the destination task.

The ability to direct a message to a parent task is very useful in background tasking. An assembler need not know from which task it was spawned and can merely direct any diagnostics to the parent task.

If the destination task number equals -1, the task message is moved to the monitor input buffer and parsed as a command line. This feature is used by the CREATE TASK BLOCK primitive to spawn a new task.

See also:

1.3.38 XGMP - GET MESSAGE POINTER
1.3.40 XGTM - GET TASK MESSAGE
1.3.44 XKTM - KILL TASK MESSAGE
1.3.90 XSMP - SEND MESSAGE POINTER
1.3.93 XSTM - SEND TASK MESSAGE

Possible Errors:

78 = Message buffer full

### 1.3.97 <u>XSTP - SET/READ TASK PRIORITY</u>

```
Mnemonic:    XSTP
Value:       $A03C
Module:      MPDOSK1
Format:      XSTP
             <status error return>

Registers: In   D0.B = Task #
                D1.W = Task time/Task priority
          Out   D1.B = Task priority (If D1.B was 0)
```

Note: If D0.B=-1, then select current task. If D1.B=0, then read task priority into D1.B.

Description: The SET/READ TASK PRIORITY primitive either sets or reads the task priority selected by data register D0.B. If D1.B is nonzero, then the priority is set. Otherwise, it is read and returned in D1.B. If the upper byte of D1.W is nonzero, then the corresponding task time slice is also set.

See also: 1.3.86 XRTS - READ TASK STATUS

Possible Errors:

74 = No such task

### 1.3.98 XSUI - SUSPEND UNTIL INTERRUPT

Mnemonic: XSUI
Value: $A01C
Module: MPDOSK1
Format: XSUI

Registers: In D1.W = EV1/EV2
Out D0.L = Event

Description: The SUSPEND UNTIL INTERRUPT primitive suspends the user task until one of the events specified in data register D1 occurs. A task can suspend until an event sets (positive event) or until it resets (negative event). A task can suspend pending two different events. This is useful when combined with timeout counters to prevent system lockups. Data register D0.L is returned with the event which caused the task to be scheduled.

A suspended task does not receive any CPU cycles until one of the event conditions is met. When the event bit is set (or reset), the task begins executing at the next instruction after the XSUI call. The task is scheduled during the normal swapping functions of PDOS according to its priority. Register D0.L is used to determined which event scheduled the task.

A suspended task is indicated in the LIST TASK (LT) command under the 'Event' parameter. Multiple events are separated by a slash.

Events 64 through 128 toggle when they cause a task to move from the suspended state to the ready state. All others must be reset by the event routine.

If a locked task attempts to suspend itself, the call polls the events until a successful return condition is met.

See also:

1.3.17 XDEV - DELAY SET/RESET EVENT
1.3.88 XSEF - SET EVENT FLAG W/SWAP
1.3.89 XSEV - SET EVENT FLAG
1.3.100 XTEF - TEST EVENT FLAG

Possible Errors: None

### 1.3.99 XSUP - ENTER SUPERVISOR MODE

Mnemonic: XSUP
Value: $A02C
Module: MPDOSK1
Format: XSUP

Registers: None

Description: The ENTER SUPERVISOR MODE primitive moves your current task from user mode to supervisor mode. Care should be taken not to crash the system since you would then be executing off the supervisor stack! This primitive enables programs to access I/O addresses and use privileged instructions.

You exit to user mode by executing a 'ANDI.W #$DFFF,SR' instruction or the XUSP primitive.

See also:
1.3.50 XLSR - LOAD STATUS REGISTER
1.3.105 XUSP - RETURN TO USER MODE

Possible Errors: None

## 1.3.100 XSWP - SWAP TO NEXT TASK

Mnemonic: XSWP
Value: $A000
Module: MPDOSK1
Format: XSWP

Registers: None

Description: The SWAP TO NEXT TASK primitive relinquishes control to the PDOS task scheduler. The next ready task with the highest priority begins executing. (This may be to the same task if there is only one task or the task is the highest priority ready task.)

Possible Errors: None

## 1.3.101 XSZF - GET DISK SIZE

```
Mnemonic:    XSZF
Value:       $A0B6
Module:      MPDOSF
Format:      XSZF
             <status error return>

Registers: In   D0.B = Disk number
           Out  D5.L = Directory size/# of files
                D6.L = Allotted/Used
                D7.L = Largest/Free
```

Description: The GET DISK SIZE primitive returns disk size parameters in data registers D5 through D7. Data register D5 returns the number of currently defined files in the low word along with the maximum number of files available in the directory in the high word.
The low order 16 bits of data register D6 (0-15) returns the total number of sectors used by all files. The high order 16 bits of D6 (16-31) returns the number of sectors allocated for file storage.

The low order 16 bits of data register D7 (0-15) is calculated from the disk sector bit map and reflects the number of sectors available for file allocation. The high order 16 bits of D7 (16-31) is returned with the size of the largest block of contiguous sectors. This is useful in defining large files.

Possible Errors:

68 = Not PDOS disk
Disk errors

## 1.3.102 XTAB - TAB TO COLUMN

Mnemonic: XTAB
Value: $A090
Module: MPDOSK2
Format: XTAB <column>

Registers: None

Description: The TAB TO COLUMN primitive positions the cursor to the column specified by the number following the call. Spaces are output until the column counter is greater than or equal to the parameter.

The first print column is zero. At least one space character will always be output.

Possible Errors: None

### 1.3.103 XTEF - TEST EVENT FLAG

```
Mnemonic:     XTEF
Value:        $A01A
Module:       MPDOSK1
Format:       XTEF
              <status return>

Registers: In   D1.B = Event number (+=0-127, -=128)
           Out    SR = NE....Event set (1)
                       EQ....Event clear (0)
```

Description: The TEST EVENT FLAG primitive sets the 68000 status word EQUAL or NOT-EQUAL depending upon the zero or nonzero state of the specified event flag. The flag is not altered by this primitive.

The event number is specified in data register D1 and is module 128. Event 128 is local to each task.

See also:

- 1.3.17 XDEV - DELAY SET/RESET EVENT
- 1.3.88 XSEF - SET EVENT FLAG W/SWAP
- 1.3.89 XSEV - SET EVENT FLAG
- 1.3.95 XSUI - SUSPEND UNTIL INTERRUPT

Possible Errors: None

## 1.3.104 XTLP - TRANSLATE LOGICAL TO PHYSICAL EVENT

```
   Mnemonic:     XTLP
   Value:        $A110
   Module:       MPDOSK1
   Format:       XTLP

Registers: In    D1.W   = Event 1.B,,Event 0.B
           Out   A0     = Event 0 address (0=no event 0 to suspend on)
                 A1     = Event 1 address (0=no event 1 to suspend on)
                 D1     = Event 1 Descriptor.w,Event 0 Descriptor.w
```

Description:

XTLP takes a VMEPROM logical event number and translates the event into a physical event. This call is used when a program needs to suspend on both a logical and a physical event. The logical event is first translated; then the XSOE call is used to suspend it.

A VMEPROM logical event is one of the 128 events maintained by the VMEPROM system in SYRAM.

Events are summarized as follows:

```
  1 - 63  = Software events
 64 - 80  = Software self clearing events
 81 - 95  = Output port events
 96 -111  = Input port events
112 -115  = Timer events
116 -127  = System control events
     128  = Local
```

The event descriptor is a 16-bit word that defines both the bit number at the specified A0,A1 address and the action to take on the bit. The following bits are defined:

```
Bit number -- 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
0
            T  x  x  x  x  x x x S x x x x B B B
```

```
   T = Should the bit be toggled on scheduling?
       1 = Yes (toggle),   0 = No (do not toggle)

   S = Suspend on event bit clear or set
       1 = Suspend on SET, 0 = Suspend on CLEAR

 BBB = The 680 x 0 bit number to use as an event
       x = Reserved, should be 0
```

Since the bit number is specified in the lower three bits of the descriptor, you may use the descriptor with the 680 x 0 BTST, BCLR, BSET instructions. You may also use the following physical manipulation calls which are macros for single assembly instructions. They are optimal as long as the values have already been placed in the correct registers. Physical events may need synchronization via the XTAS macro to avoid corruption. The macros are defined in the file PESMACS:SR.

XTST -☺Test Physical Event (replaces BTST D1, A0))
XSET - Test and Set Physical Event (replaces BSET D1,(A0))
XCLR - Test and Clear Physical Event (replaces BCLR D1,(A0))

```
Input:          D1.W - Event descriptor
                A0   - Event address
Output:   None
                Status:   EQ - the bit was clear (0)
                          NE - the  bit was set (1)
```

The bottom three bits are evaluated as a bit number. The bit at the address is set and the previous value is returned in the Z bit of the status register.

XTAS - Test and Set Physical Event (Bit 7 atomic)

This macro replaces TAS (A0). The seventh bit at the address is set and the previous value is returned in the N bit of the status register.

```
Input:          A0 - Event  address
Output:   None
Status:   EQ -  the bit was clear (0)
                NE -  the bit was set (1)
```

```
See also:       XDPE - Delay On Physical Event
                     XSOE - Suspend On Physical Event
```

### 1.3.105 XUAD - UNPACK ASCII DATE

```
Mnemonic:    XUAD
Value:       $A036
Module:      MPDOSK3
Format:      XUAD

Registers: In   D1.W = (Year*16+Month)*32+Day
                       (YYYY YYYM MMMD DDDD)
           Out  (A1) = 'DY-MON-YR'<null>
                       (Outputs ??? for invalid months)
```

Description: The UNPACK ASCII DATE primitive returns a pointer in address register A1 to an ASCII date string. Data register D1.W contains the binary date [(Year*16+Month)*32+Day]. The format of the string is more exact than simple numbers separated by slashed.

Note: XUAD does not check for a valid date and hence, funny looking strings could result. Invalid months are replaced by '???.'

See also:

```
        1.3.28  XFTD - FIX TIME & DATE
        1.3.52  XPAD - PACK ASCII DATE
        1.3.71  XRDT - READ DATE
        1.3.84  XRTM - READ TIME
        1.3.102 XUDT - UNPACK DATE
        1.3.106 XUTM - UNPACK TIME
```

Possible Errors: None

## 1.3.106 XUDT - UNPACK DATE

```
Mnemonic:    XUDT
Value:       $A060
Module:      MPDOSK3
Format:      XUDT

Registers: In   D1.W = (Year * 16 + Month) * 32 + Day
           Out  (A1) = 'MN/DY/YR'<null>
```

Description: The UNPACK DATE primitive converts a one-word encoded date into an eight- character string terminated by a null (nine characters total). Data register D1 contains the encoded date and returns with a pointer to the formatted string in address register A1. The output of the FIX TIME & DATE (XFTD) primitive is valid input to this primitive.

See also:

- 1.3.28 XFTD - FIX TIME & DATE
- 1.3.52 XPAD - PACK ASCII DATE
- 1.3.71 XRDT - READ DATE
- 1.3.84 XRTM - READ TIME
- 1.3.101 XUAD - UNPACK ASCII DATE
- 1.3.106 XUTM - UNPACK TIME

Possible Errors: None

### 1.3.107 XULF - UNLOCK FILE

```
Mnemonic:    XULF
Value:       $A0EE
Module:      MPDOSF
Format:      XULF
             <status error return>

Registers: In   D1.W = File ID
```

Description: The UNLOCK FILE primitive unlocks a locked file for access by any other task. The file is specified by the file ID in data register D1.

See also: 1.3.48 XLKF - LOCK FILE

Possible Errors:

```
        52 = File not open
        59 = Invalid slot #
        Disk errors
```

### 1.3.108 XULT - UNLOCK TASK

Mnemonic: XULT
Value: $A016
Module: MPDOSK1
Format: XULT

Registers: None

Description: The UNLOCK TASK primitive unlocks the current task by clearing the swap lock variable in system RAM. This allows other tasks to be scheduled and receive CPU time.

See also:
1.3.49 XLKT - LOCK TASK

Possible Errors: None

### 1.3.109 **XUSP - RETURN TO USER MODE**

Mnemonic: XUSP
Value: $A008
Module: MPDOSK1
Format: XUSP

Registers: None

Description: The RETURN TO USER MODE primitive moves your current task from supervisor mode to user mode. Executing an 'ANDI.W #$DFFF,SR'' instruction also returns you to user mode, but must be executed in supervisor mode. The XUSP primitive can be executed in either mode.

See also:
1.3.50 XLSR - LOAD STATUS REGISTER
1.3.96 XSUP - ENTER SUPERVISOR MODE

Possible errors: None

## 1.3.110 XUTM - UNPACK TIME

Mnemonic: XUTM
Value: $A062
Module: MPDOSK3
Format: XUTM

Registers: In D1.W = HOUR*256+MINUTE
(HHHH HHHH MMMM MMMM)
Out (A1) = HR:MN<null>

Description: The UNPACK TIME primitive converts a one word encoded date into a five character string terminated by a null (six characters total). Data register D1 contains the encoded time and returns a pointer to the formatted string in address register A1. The output of the FIX TIME & DATE (XFTD) primitive is valid input to this primitive.

See also:

- 1.3.28 XFTD - FIX TIME & DATE
- 1.3.52 XPAD - PACK ASCII DATE
- 1.3.71 XRDT - READ DATE
- 1.3.84 XRTM - READ TIME
- 1.3.101 XUAD - UNPACK ASCII DATE
- 1.3.102 XUDT - UNPACK DATE

Possible Errors: None

### 1.3.111 XVEC - SET/READ EXCEPTION VECTOR

```
Mnemonic:    XVEC
Value:       $A116
Module:      MPDOSK1
Format:      XVEC

Registers: In   D0.W  = Exception number (#2-255)
                (A0)    =  New  exception  service  routine
(0=read
                        only)
           Out  (A0)  = Old service routine
```

Description: XVEC sets and/or reads the execution vector for the system. The old service routine address is returned so that you may change a routine and then restore the former routine under program control.

See also: XDTV - Define Trap Vectors

Possible Errors: None

### 1.3.112 <u>XWBF - WRITE BYTES TO FILE</u>

```
Mnemonic:     XWBF
Value:        $A0F0
Module:       MPDOSF
Format:       XWBF
              <status error return>

Registers: In   D0.L = Byte count - must be positive
                D1.W = File ID
                (A2) = Buffer address
```

Description: The WRITE BYTES TO FILE primitive writes from a memory buffer, pointed to by address register A2, to a disk file specified by the file ID in register D1. Register D0 specifies the number of bytes to be written. If the channel buffer has been rolled to disk, the least-used buffer is freed and the buffer is restored to memory. The file slot ID is placed on the top of the last-access queue.

The write is independent of the data content. The buffer pointer in register A2 may be on any byte boundary. The write operation is not terminated with a null character.

A byte count of zero in register D0 results in no data being written to the file.

If it is necessary for the file to be extended, PDOS first uses sectors already linked to the file. If a null or end link is found, a new sector obtained from the disk sector bit map is linked to the end of the file. If this makes the file non-contiguous, it is retyped as a non-contiguous file.

See also:

1.3.65 XRBF - READ BYTES FROM FILE
1.3.74 XRLF - READ LINE FROM FILE
1.3.111 XWLF - WRITE LINE TO FILE

Possible Errors:

52 = File not open
58 = File delete or write protected
59 = Invalid slot #
60 = File space full
Disk errors

## 1.3.113 XWDT - WRITE DATE

Mnemonic: XWDT
Value: $A064
Module: MPDOSK3
Format: XWDT

Registers: In D0.B = Month (1-12)
D1.B = Day (1-31)
D2.B = Year (0-99)

Description: The WRITE DATE primitive sets the system date counters. Register D0 specifies the month and ranges from 1 to 12. Register D1 specifies the day of month and ranges from 1 to 31. Register D2 is the last 2 digits of the year.

No check is made for a valid date.

Possible Errors: None

## 1.3.114 <u>XWFA - WRITE FILE ATTRIBUTES</u>

```
Mnemonic:    XWFA
Value:       $A0F2
Module:      MPDOSF
Format:      XWFA
             <status error return>

Registers: In   (A1) = File name
                (A2) = ASCII file attributes
```

Note: (A2)=0 clears all attributes.

Description: The WRITE FILE ATTRIBUTES primitive sets the attributes of the file specified by the file name pointed to by register A1. Register A2 points to an ASCII string containing the new file attributes followed by a null character. The format is:

```
(A2) = {file type}{protection}

{file type}  = AC - Procedure file
               BN - Binary file
               OB - 68000 object file
                    SY - 68000 memory image
               BX - BASIC binary token file
               EX - BASIC ASCII file
               TX - Text file
               DR - System I/O driver

{protection} = *  - Delete protect
               ** - Delete and Write protect
```

If register A2 points to a zero byte, then all flags, with the exception of the contiguous flag, are cleared.

See also:

1.3.11 XCFA - CLOSE FILE W/ATTRIBUTE
1.3.72 XRFA - READ FILE ATTRIBUTES
1.3.110 XWFP - WRITE FILE PARAMETERS

Possible Errors:

50 = Invalid file name
53 = File not defined
54 = Invalid file type
Disk errors

### 1.3.115 XWFP - WRITE FILE PARAMETERS

Mnemonic: XWFP
Value: $A0FC
Module: MPDOSF
Format: XWFP
<status error return>

Registers: In (A1) = File name
D0.L = Sector index of EOF/Bytes in last sector
D1.L = Time/Date created
D2.L = Time/Date last accessed
D3.W = OR'd status (less contiguous bit)

Description: The WRITE FILE PARAMETERS primitive updates the end-of-file and date parameters of the file specified by the name pointed to by address register A1 in the disk directory.

See also:
1.3.11 XCFA - CLOSE FILE W/ATTRIBUTE
1.3.72 XRFA - READ FILE ATTRIBUTES
1.3.109 XWFA - WRITE FILE ATTRIBUTES

Possible Errors:

50 = Invalid file name
53 = File not defined
Disk errors

### 1.3.116 XWLF - WRITE LINE TO FILE

Mnemonic: XWLF
Value: $A0F4
Module: MPDOSF
Format: XWLF
<status error return>

Registers: In D1.W = File ID
(A2) = Buffer address

Description: The WRITE LINE TO FILE primitive writes a line delimited by a null character to the disk file specified by the file ID in register D1. Address register A2 points to the string to be written. If the channel buffer has been rolled to disk, the least-used buffer is freed and the buffer is restored to memory. The file slot ID is placed on the top of the last-access queue.

The write line primitive is independent of the data content, with the exception that a null character terminates the string. The buffer pointer in register A2 may be on any byte boundary. A single write operation continues until a null character is found.

If it is necessary for the file to be extended, PDOS first uses sectors already linked to the file. If a null link is found, a new sector obtained from the disk sector bit map is linked to the end of the file. If this makes the file non-contiguous, it is retyped as a non-contiguous file.

See also: 1.3.65 XRBF - READ BYTES FROM FILE
1.3.74 XRLF - READ LINE FROM FILE
1.3.107 XWBF - WRITE BYTES TO FILE

Possible Errors:

52 = File not open
58 = File delete or write protected
59 = Invalid slot #
60 = File space full
Disk errors

### 1.3.117 **XWSE - WRITE SECTOR**

Mnemonic: XWSE
Value: $A0C6
Module: MPDOSF
Format: XWSE
<status error return>

Registers: In D0.B = Disk number
D1.W = Sector number
(A2) = Buffer address

Description: The WRITE SECTOR primitive is a system-defined, hardware-dependent program which writes 256 bytes of data from a buffer, pointed to by address register A2, to the logical sector and disk device specified by data registers D1 and D0 respectively.

See also:

CHAPTER 8 BIOS
1.3.42 XISE - INITIALIZE SECTOR
1.3.79 XRSE - READ SECTOR
1.3.82 XRSZ - READ SECTOR ZERO

Possible Errors:

Disk errors

### 1.3.118 XWTM - WRITE TIME

Mnemonic: XWTM
Value: $A066
Module: MPDOSK3
Format: XWTM

Registers: In D0.B = Hours (0-23)
D1.B = Minutes (0-59)
D2.B = Seconds (0-60)

Description: The WRITE TIME primitive sets the system clock time. Register D0 specifies the hour and ranges from 0 to 23. Register D1 specifies the minutes and register D2, the seconds. The latter two range from 0 to 59.

There is no check made for a valid time.

Possible Errors: None

## 1.3.119 XZFL - ZERO FILE

```
Mnemonic:    XZFL
Value:       $A0F6
Module:      MPDOSF
Format:      XZFL
             <status error return>

Registers: In   (A1) = File name
```

Description: The ZERO FILE primitive clears a file of any data. If the file is defined, then the end-of-file marker is placed at the beginning of the file. If the file is not defined, it is defined with no data.

See also:

1.3.18 XDFL - DEFINE FILE
1.3.19 XDLF - DELETE FILE

Possible errors:

50 = Invalid file name
61 = File already open
68 = Not PDOS disk
Disk errors

**APPENDIX TO THE**
**VMEPROM USER'S MANUAL**

# TABLE OF CONTENTS

# A P P E N D I X   A

## VMEPROM ERROR DEFINITIONS

The error numbers 1 - 49 are errors detected by the VMEPROM command interpreter. The error numbers 50 - 100 are kernel or file manager detected errors and the numbers 100-200 are disk I/O errors.

### A.1 VMEPROM Command Errors

**ERROR 1** Syntax error. This error is noted whenever a incorrect number of parameters are specified with a VMEPROM command line.

**ERROR 2** Command line argument error. This error is noted whenever a incorrect parameter is detected on a VMEPROM command line.

**ERROR 3** Illegal Register. Illegal register name specified.

**ERROR 4** Cannot open file. If an attempt was made to open a nonexistent file with a VMEPROM command.

**ERROR 5** Checksum error. Checksum error during loading of S-records.

**ERROR 6** No open AC file

**ERROR 7** Breakpoint number out of range. An illegal breakpoint was specified.

**ERROR 8** No such breakpoint. A not existing breakpoint was tried to replace or delete.

**ERROR 9** Not used.

**ERROR 10** No Floating point Coprocessor. An attempt was made to modify a Coprocessor register while no coprocessor is available.

**ERROR 11** Illegal driver. There is no right driver ID.

**ERROR 12** Driver list is full.

**ERROR 13** Controller not in system. Try CONFIG.

**ERROR 14** UART type/utility already installed

**ERROR 15** Begin address is greater than end address

**ERROR 16** Illegal odd address

**ERROR 17** RAM-disk table is full

**ERROR 18** All name buffers are full

**ERROR 19-49** Not used

## A.2 <u>Kernel and File Manager Errors</u>

**ERROR 50** ILLEGAL FILE NAME. Valid file names consist of an alpha character followed by up to 7 alphanumeric characters. An optional extension and disk number may follow. An extension consists of a colon followed by 1 to 3 characters. A disk number consists of a slash and a number ranging from 0 to 127.

**ERROR 51** FILE ALREADY DEFINED. Each file name is unique to a disk file directory. There is one directory per disk number.

**ERROR 52** FILE NOT OPEN. An attempt to access a file which has not been opened, results in error 52.

**ERROR 53** FILE NOT DEFINED. If the file name does not exist in the disk directory, an error 53 occurs.

**ERROR 54** INVALID FILE TYPE. Valid file types are AC, BN, OB, SY, BX, EX, TX, DR, *, and **. All others result in error.

**ERROR 55** FRAGMENTED. Error 55 results from attempting to define a contiguous file on a disk unit which does not have enough room or is fragmented such that there is not a big enough contiguous block of sectors.

**ERROR 56** END-OF-FILE. Error 56 comes from an attempt to read past the END-OF-FILE index of a file.

**ERROR 57** DIRECTORY FULL. The file directory size is set when the file is initialized. Any attempt to define another file after the directory has been filled, results in error 57.

**ERROR 58** FILE DELETE PROTECTED. An attempt to delete a file with a delete or write protect flag results in error 58.

**ERROR 59** INVALID SLOT #. A valid file slot number is returned from the file manager on all open commands. A file slot consists of the disk number in the left byte and the slot index in the right byte.

**ERROR 60** DISK SPACE FULL. An attempt to extend a file or define a file after the disk space is filled results in error 60.

**ERROR 61** FILE ALREADY OPEN. A file can be opened only once in sequential (XSOP) and random (XROP) modes. Read only open (XROO) and shared random open (XNOP) can be executed more than once on the same file.

**ERROR 62** NO START ADDRESS. An object (OB) file must have a start address. This is generated by an address parameter for the 'END' statement in the assembly source.

**ERROR 63** ILLEGAL OBJECT TAG. Only hex object tag characters are legal.

**ERROR 64** ILLEGAL SECTION. Only section 0 is executable under VMEPROM.

**ERROR 65** FILE NOT LOADABLE. Only files typed 'OB', 'SY' are loadable by VMEPROM.

**ERROR 66** ILLEGAL PORT NUMBER OR BAUD RATE. Only 1 through 15 are legal ports. Valid baud rates are 110, 300, 600, 1200, 2400, 4800, 9600, and 19200.

**ERROR 67** INVALID PARAMETER. Most monitor commands check parameters for valid ranges and types.

**ERROR 68** NOT A PDOS DISK. An initialized PDOS disk has the constant >A55A at location >0028 of the header sector (sector 0). If the constant is not found on a disk read, error 68 results.

**ERROR 69** NOT ENOUGH FILE SLOTS. A maximum of 64 files can be open at a time. These correspond to the 64 file slots.

**ERROR 70** POSITION ERROR. Error 70 results from a position command beyond the end-of-file index.

**ERROR 71** NESTING ERROR. Error 71 results for nesting procedure files too deep.

**ERROR 72** TOO MANY TASKS. A maximum of 64 tasks are supported by VMEPROM.

**ERROR 73** NOT ENOUGH MEMORY. An attempt to create a task with more memory than the current task or available memory in the system memory bit maps, results in error 73.

**ERROR 74** NO SUCH TASK. Error 74 occurs when referencing a task not in the task list or task 0.

**ERROR 75** FILE LOCKED. Once a file has been locked (XLKF), it cannot be accessed until unlocked (XULF).

**ERROR 76** TASK LOCKED. Once a task has been locked (XLKT), it cannot be killed until unlocked (XULT).

**ERROR 77** Not used.

**ERROR 78** MESSAGE BUFFER FULL. There are 64 message buffers in the supported by VMEPROM. Too many messages results in error 78.

**ERROR 79** MEMORY ERROR. Error results from a XFUM primitive with invalid arguments.

**ERROR 80** I/O DRIVER ERROR. Driver dependent.

**ERROR 81** UNIMPLEMENTED PDOS PRIMITIVE. A defined PDOS primitive is not currently implemented.

**ERROR 82** ILLEGAL PDOS PRIMITIVE. An invalid A-line primitive has been executed.

**ERROR 83** DELAY EVENT STACK FULL. Too many delayed events have been requested.

**ERROR 84** CHECKSUM ERROR. Not implemented.

**ERROR 85** ABORTED TASK. If a task is aborted by the scheduler, error 85 results.

**ERROR 86** PHANTOM PORT. A task has made a call to get character without any possibility of getting a character.

## A.3 Disk Errors

**COMMON ERRORS**

| | |
|---|---|
| **ERROR 100** | Illegal drive |
| **ERROR 101** | Sector too big |
| **ERROR 102** | Timeout |

**WFC-1 ERRORS**

| | |
|---|---|
| **ERROR 103** | Write protected |
| **ERROR 104** | Address Mark not found |
| **ERROR 105** | No Track 0 |
| **ERROR 106** | Not ready |
| **ERROR 107** | DMA error |
| **ERROR 108** | Sector not found |
| **ERROR 109** | ID read error |
| **ERROR 110** | Uncorrectable data |
| **ERROR 111** | Bad block |
| **ERROR 112-119** | NOT USED |
| **ERROR 120** | ILLEGAL DRIVER. Driver installed? |
| **ERROR 121-146** | NOT USED |

**SCSI Errors**

| | |
|---|---|
| **ERROR 147** | SCSI Parity error |
| **ERROR 148-152** | NOT USED |
| **ERROR 153** | Illegal Command |
| **ERROR 154-156** | NOT USED |
| **ERROR 157** | Address Error |
| **ERROR 158** | ISCSI-1 Exception |
| **ERROR 159** | Target Mode |
| **ERROR 160** | NOT USED |
| **ERROR 161** | SCSI timeout |
| **ERROR 162** | LUN not ready |
| **ERROR 163** | SCSI read error |

| | |
|---|---|
| **ERROR 164** | SCSI write error |
| **ERROR 165** | Block not found |
| **ERROR 166** | Target reservation conflict |
| **ERROR 167** | Format error |
| **ERROR 168** | NOT USED |
| **ERROR 169** | Check Condition Status |
| **ERROR 170-171** | NOT USED |
| **ERROR 172** | Floppy not ready |
| **ERROR 173** | FDC read error |
| **ERROR 174** | FDC write error |
| **ERROR 175** | FDC record not found |
| **ERROR 176** | Floppy write protected |
| **ERROR 177** | FDC format error |

# A P P E N D I X B

## VMEPROM DISK LAYOUT

The following disk sector listings define the VMEPROM disk formats including the header sector, directory entries, and data storage.

### B.1 <u>Disk Header Sector Contents</u>

```
000-00F  56 4D 45 50 52 4F 4D 00 00 00 00 00 00 00 00 00 VMEPROM.........
010-01F  00 00 01 22 00 00 00 00 04 00 BA A0 A5 5A FF FF ..."......: %Z..
020-02F  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
030-03F  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
040-04F  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
050-05F  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
060-06F  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
070-07F  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
080-08F  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
090-09F  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
0A0-0AF  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
0B0-0BF  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
0C0-0CF  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
0D0-0DF  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
0E0-0EF  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
0F0-0FF  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
```

```
$00-$0F    VMEPROM  = Disk name
$10-$11    0000     = Boot sector (None)
$12-$13    0122     = # of files
$14        00       = # of boot sectors (None)
$15-$17    000000   = Boot address (None)
$18-$19    0400     = Maximum # of files
$1A-$1B    BAA0     = Maximum # of sectors
$1C-$1D    A55A     = PDOS ID
$1E-$1F    FFFF     = Sides/Density
$20-                = Sector allocation map (0=free, 1=used)
```

## B.2 Directory Contents

```
000-00F  41 4D 41 5A 49 4E 47 00 00 00 00 05 08 00 00 12 AMAZING.........
010-01F  00 00 00 12 00 12 00 9A 10 1F A8 A2 10 1F A8 A2 ..........("..("
020-02F  41 53 4D 00 00 00 00 00 00 00 00 00 80 00 00 25 ASM............%
030-03F  00 00 00 00 00 00 00 2E 10 1F A8 A2 10 1F A8 A2 ..........("..("
040-04F  42 30 31 00 00 00 00 00 00 00 00 0A 20 00 00 26 B01......... ..&
050-05F  00 00 00 01 00 01 00 58 10 1F A8 A2 10 1F A8 A2 .......X..("..("
060-06F  42 30 31 00 00 00 00 00 53 52 00 0A 02 00 00 28 B01.....SR.....(
070-07F  00 00 00 04 00 04 00 55 10 1F A8 A2 10 1F A8 A2 .......U..("..("
080-08F  42 30 32 00 00 00 00 00 00 00 00 0A 20 00 00 2D B02......... ..-
090-09F  00 00 00 01 00 01 00 5B 10 1F A8 A2 10 1F A8 A2 .......[..("..("
0A0-0AF  42 30 32 00 00 00 00 00 53 52 00 0A 02 00 00 2F B02.....SR...../
0B0-0BF  00 00 00 04 00 04 00 3D 10 1F A8 A2 10 1F A8 A2 .......=..("..("
0C0-0CF  42 30 33 00 00 00 00 00 00 00 00 0A 20 00 00 34 B03......... ..4
0D0-0DF  00 00 00 01 00 01 00 5B 10 1F A8 A2 10 1F A8 A2 .......[..("..("
0E0-0EF  42 30 33 00 00 00 00 00 53 52 00 0A 02 00 00 36 B03.....SR.....6
0F0-0FF  00 00 00 04 00 04 00 3F 10 1F A8 A2 10 1F A8 A2 .......?..("..("
```

```
$00 - $07   = File name
$08 - $0A   = File extension
$0B         = Directory level
$0C - $0D   = Type
$0E - $0F   = Start sector
$10 - $11   = Free
$12 - $13   = Sectors allocated
$14 - $15   = EOF sector index
$16 - $17   = # of bytes in last sector
$18 - $1B   = Date created
$1C - $1F   = Date last updated
$20           Next directory entry
```

### B.3 Data Sectors

```
000-00F  00 A1 00 9F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F .!..____________
010-01F  5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F ________________
020-02F  5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F ________________
030-03F  5F 5F 5F 5F 0D 0D 4D 65 6D 6F 72 79 20 6D 6F 64 ____..Memory mod
040-04F  69 66 79 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F ify_____________
050-05F  5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F ________________
060-06F  5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F ________________
070-07F  5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F ________________
080-08F  5F 5F 5F 5F 5F 0D 0D 4D 6F 76 65 20 6D 65 6D 6F _____..Move memo
090-09F  72 79 20 63 6F 6E 74 65 6E 74 73 5F 5F 5F 5F 5F ry contents_____
0A0-0AF  5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F ________________
0B0-0BF  5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F ________________
0C0-0CF  5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F ________________
0D0-0DF  5F 5F 5F 5F 5F 5F 0D 0D 53 65 74 20 6D 65 6D 6F ______..Set memo
0E0-0EF   72 79 20 74 6F 20 63 6F 6E 73 74 61 6E 74 20 76 ry to constant
v
0F0-0FF  61 6C 75 65 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F 5F alue____________
```

```
$00 - $01  = Forward link
$02 - $03  = Backward link
$04 - $FF  = Data storage
```

**This page was intentionally left blank**

# APPENDIX C

# LOADABLE I/O DRIVERS

## C.1 General

The VMEPROM I/O drivers are an extension of the file system. If a file's attribute is 'DR', then the file manager expects the file to be an I/O driver program instead of data. I/O driver files contain position independent (self-relocating) code.

When an I/O driver is opened, closed, read from, written to, or positioned, the file manager branches into the channel buffer at specific entry points. This requires that the first twelve bytes of the file be reserved for branch instructions and that the driver code and variables be no more than 240 bytes in length.

The following driver entry points must be at the beginning of each driver module:

Driver entry points:

```
        DC.W    $A55B           ;DRIVER ID
DROP    BRA.S   OPEN            ; 2 OPEN
DRCL    BRA.S   CLOS            ; 4 CLOSE
DRRD    BRA.S   READ            ; 6 READ
DRWR    BRA.S   WRIT            ; 8 WRITE
DRPS    BRS.S   POST            ;10 POSITION
```

The driver must be written in position independent or self-relocating 68000 assembly code. This simply means that while the code is relocatable, there can be no relocatable tags within the object file.

A common way to make the code self-relocating is to generate a base address and then reference each constant within the program as a displacement beyond the base address. PDOS passes the base address of the driver buffer in address register A2. This can be conveniently used as the base register for variables defined as the label minus the start address plus four. The former makes the label absolute (relocatable-relocatable=absolute) and the latter skips the file links.

The PDOS file manager passes all parameters in registers to I/O drivers. All registers are available for use by the driver except address registers A4 through A7.

The driver executes in supervisor mode. The return address is already on the system stack. The status register passes the error conditions back to the PDOS file manager. An 'EQ' status indicates that no error occurred. A 'NE' status specifies an error with the error number returned in data register D0.

The data and address registers of the file manager call are located on the stack immediately following the return address, where D0 is 4(A7), D1 is 8(A7), and so on. This is useful for passing the number of bytes on the end of file to the D3.L of the file manager call. See the input driver example E.6.

If the driver alters constants within the buffer, then the file altered bit must be set in the file slot so that the buffer is correctly restored when rolled to the disk.  This is done by executing the instruction 'ORI.W #$8000,12(A4)' or 'TAS.B 12 (A4)'.

The following table describes the register usage for each driver entry point:

```
      OPEN:  D7.W = Channel status
             (A2) = Driver base + 4
             (A4) = File slot
             (A5) = SYSRAM
             (A6) = Task TCB
             (A7) = Return address

     CLOSE:  D7.W = Channel status
             (A2) = Driver base + 4
             (A4) = File slot
             (A5) = SYSRAM
             (A6) = Task TCB
             (A7) = Return address

      READ:  D5.L = Character count (-1 = Line operation)
             D7.W = Channel status
             (A2) = Driver base + 4
             (A3) = Memory buffer
             (A4) = File slot
             (A5) = SYSRAM
             (A6) = Task TCB
             (A7) = Return address
        3*4+4(A7) = Return EOF bytes to D3.L


     WRITE:  D5.L = Character count (-1 = Line operation)
             D7.W = Channel status
             (A2) = Driver base + 4
             (A3) = Memory buffer
             (A4) = File slot
             (A5) = SYSRAM
             (A6) = Task TCB
             (A7) = Return address

  POSITION:  D5.L = Character position
             D7.W = Channel status
             (A2) = Driver base + 4
             (A4) = File slot
             (A5) = SYSRAM
             (A6) = Task TCB
             (A7) = Return address
```

## C.2 RESTRICTIONS

The following summarizes the restrictions when adding an I/O driver to VMEPROM:

1) Drivers must be written in self-relocating, address independent 68000 assembly language.

2) The driver identification constant $A55B must be the first word of the driver.

3) Driver entry points must immediately follow the driver identification word.

4) An I/O driver code and variables cannot exceed the sector size less four link bytes. This results in a maximum length of 252 bytes.

5) A driver MUST NOT make any console or file I/O system calls.

6) A driver is exited via an 'RTS' instruction. A 'NE' status condition indicates a driver error with data register D0 passing the error number.

7) Larger drivers can be written, but the excess code must be located elsewhere in memory.

8) Drivers execute in supervisor mode.

9) Address registers A4, A5, A6, and A7 must be preserved.

## C.3 Output Driver Example

The following program is an example of a VMEPROM I/O driver. The output is to the logical port number found in the TCB variable U1P$.

```
*   TTO:SR         10/02/87
**********************************************************
*                                                        *
*       66   888  K   K    PPPP  DDDD   OOO   SSS        *
*      6    8   8 K  K     P   P  D  D O   O S   S       *
*     6     8   8 K K      P   P  D  D O   O S           *
*     6666   888  KK       PPPP   D  D O   O  SSS        *
*     6   6 8   8 K K      P      D  D O   O     S       *
*     6   6 8   8 K  K     P      D  D O   O S   S       *
*      666   888  K   K    P     DDDD   OOO   SSS        *
*                                                        *
* TTTTT TTTTT  OOO     DDDD  RRRR  III V   V EEEEE RRRR  *
*   T     T   O   O     D  D R   R  I  V   V E     R   R *
*   T     T   O   O     D  D R   R  I  V   V E     R   R *
*   T     T   O   O     D  D RRRR   I   V V  EEEE  RRRR  *
*   T     T   O   O     D  D R R    I   V V  E     R R   *
*   T     T   O   O     D  D R  R   I    V   E     R  R  *
*   T     T    OOO     DDDD  R   R III   V   EEEEE R   R *
*                                                        *
**********************************************************
*   Eyring Research Inst.  Copyright 1983,1987.
*   ALL RIGHTS RESERVED.
*
*        Module Name: TTO
*             Author: Paul Roper
*   Revision History:
*
*   10/02/87 3.3   Update to 3.3 $A55B
*   06/20/86 3.0   Fixed for upper D1.L=output event #
                    for printers
*   02/11/86 2.0   Fixed XON/XOFF look before calling put
*
TTO IDNT 3.3  68K PDOS TTO DRIVER
*
**********************************************************
*
* This driver is intended to  output  files  to the terminal.
* It outputs file data  to  Unit 1 Port  (U1P$)  of  the task
* that opened it.   It  filters the output stream by ignoring
* <LF>, converting <CR> characters to <CR><LF> pairs, keeping
* an independent column counter and expanding <TAB> to column
* positions (multiples of 8), using blanks.   <BS>  backspace
* characters decrement the counter.  Output events, XON/XOFF,
* and DTR line checks are all supported.
*
* D5.L = Character count (-1 = Line)
* D7.W = Channel status
* (A2) = Driver base + 4
* (A3) = Memory buffer
* (A4) = File slot
* (A5) = SYSRAM
* (A6) = Task TCB
* (A7) = Return address
*
```

```
OPT PDOS,CRE
BURT     EQU      $007C                          ;BIOS UART TBL
*
SECTION  0
DTTO     DC.W     $A55A+1                        ;DRIVER ID
DROP     BRA.S    OPEN                           ; 2 OPEN
DRCL     BRA.S    CLOS                           ; 4 CLOSE
DRRD     BRA.S    READ                           ; 6 READ
DRWR     BRA.S    WRIT                           ; 8 WRITE
DRPS     MOVEQ.L  #70,D0                         ;10 POSITION ERROR
         RTS
*
READ     MOVEQ.L  #80,D0                         ;ERROR  80,  DRIVER
                                       ERROR
         RTS
*
OPEN     ORI.W    #$8000,12(A4)                  ;FILE ALTERED
         CLR.B    CCNT(A2)             ;CLEAR COUNTER
         CLR.W    D1                   ;D1=PORT #
         MOVE.B   U1P$(A6),D1          ;D1=PORT #
         MOVEQ.L  #80,D3
         ADD.B    D1,D3
         MOVE.W   D3,OUTE(A2)          ;D3=OUTPUT EVENT #
         MOVE.B   UTYP.(A5,D1.W),D3    ;D3=UART TYPE
         MOVE.B   D3,TYPE(A2)          ;SAVE FOR FUTURE
         LSL.W    #2,D3                ;POINT TO DSR
         MOVEA.L  (A5),A0
         ADDA.L   BURT(A0,D3.W),A0
         ADDQ.W   #2,A0                ;A0=PUTC ENTRY
         MOVE.L   A0,PUTC(A2)          ;SAVE PUTC ADR
         LSL.W    #2,D1                ;SAVE BASE ADR
         LEA.L    UART.(A5),A0
         MOVE.L   0(A0,D1.W),PADR(A2)
         LSR.W    #2,D1                ;SAVE FLAGS
         PEA      F8BT.(A5,D1.W)       ;PUSH POINTER TO FLAGS
         MOVE.L   (A7)+,FADR(A2)       ;SAVE PTR
*
CLOS     CLR.W    D0                             ;RETURN .EQ.
         RTS
*
*****************************************
*   WRITE CHARACTERS
*
WRIT          ORI.W     #$8000,12(A4)       ;N, ALTERED
*
WRIT02        MOVEQ.L   #0,D0               ;GET CHARACTER
              MOVE.B    (A3)+,D0            ;DONE?
              BNE.S     WRIT04              ;N
              TST.L     D5                  ;Y, WRITE LINE?
              BMI.S     CLOS                ;Y, DONE
*
WRIT04        CMPI.B    #$08,D0             ;BACKSPACE?
              BNE.S     WRIT06              ;N
              SUBQ.B    #1,CCNT(A2)         ;Y
*

WRIT06        CMPI.B    #$09,D0             ;OK, TAB?
              BNE.S     WRIT08              ;N
              MOVEQ.L   #' ',D0             ;Y
              MOVEQ.L   #7,D1               ;GET MASK
```

```
              AND.B     CCNT(A2),D1          ;GET COUNTER
              SUBQ.B    #7,D1           ;TAB BOUNDARY?
              BEQ.S     WRIT08          ;Y
              SUBQ.W    #1,A3           ;N, DO AGAIN
              TST.L     D5              ;WRITE LINE?
              BMI.S     WRIT08          ;Y
              ADDQ.L    #1,D5           ;N, BACKUP
*
WRIT08        CMPI.B    #$0A,D0         ;LF?
              BEQ.S     WRIT16          ;Y, IGNORE
              CMPI.B    #$0D,D0         ;N, CR?
              BNE.S     WRIT10          ;N
              CLR.B     CCNT(A2)        ;Y, CLEAR CCNT
              MOVE.W    #$0A0D,D0       ;CHANGE TO CRLF
*
WRIT10        CMPI.B    #' ',D0         ;CONTROL?
              BLT.S     WRIT12          ;Y
              ADDQ.B    #1,CCNT(A2)     ;N, UP COUNT
*
WRIT12        TST.B     TYPE(A2)        ;DEFINED TYPE?
              BEQ.S     CLOS            ;N, SKIP IT
              MOVE.L    OUTE(A2),D1     ;GET OUT EFVENT TO UPPER WORD
                                         ;OF D1
              MOVEA.L   FADR(A2),A0     ;GET PTR TO FLGS
              MOVE.B    (A0),D1         ;TEST FLAG EACH TIME
              BTST.L    #0,D1           ;^S^Q CHECK?
              BEQ.S     WRIT14          ;N
              TST.B     D1              ;Y, ^S STOP SET?
              BMI.S     WRIT12          ;Y, WAIT HERE
*
WRIT14        MOVEA.L   PADR(A2),A0     ;UART BASE ADR
              DC.W $4EB9,0,0            ;JSR PUTC.L
              PUTC EQU  *-DTTO          ;RETRY?
              BNE.S     WRIT12          ;Y
              LSR.W     #8,D0           ;N, 2 CHARS?
              BNE.S     WRIT12          ;Y
*
WRIT16        SUBQ.L    #1,D5           ;DONE?
              BNE.S     WRIT02          ;N
*             BRA       CLOS2           ;Y
              RTS                       ;Y, RETURN .EQ.
*
*****************************************
*   DRIVER VARIABLES
*
    OFFSET    *-DTTO+4
PADR     DC.L 0          ;BASE ADR
FADR     DC.L 0          ;UART FLAGS ADDRESS
OUTE     DC.W 0          ;OUTPUT EVENT #
CCNT     DC.B 0          ;COLUMN COUNT
TYPE     DC.B 0          ;PORT TYPE
    EVEN




*
```

```
****************************************
*   DRIVER LENGTH CHECK
*
    IFLT 256-(TYPE+1)
    FAIL ** DRIVER LENGTH ERROR! **
    ENDC
*
    END  DTTO
```

## C.4 <u>Input Driver Example</u>

The following program is an example of a VMEPROM I/O driver. The input is from the logical port number found in the TCB variable PRT$.

```
*   TTI:SR         10/02/87
*******************************************************************
*                                                                *
*          66   888  K   K    PPPP  DDDD   OOO   SSS             *
*         6    8   8 K  K     P   P  D  D O   O S   S            *
*        6     8   8 K K      P   P  D  D O   O S                *
*        6666   888  KK       PPPP   D  D O   O  SSS             *
*        6   6 8   8 K K      P      D  D O   O     S            *
*        6   6 8   8 K  K     P      D  D O   O S   S            *
*         666   888  K   K    P     DDDD   OOO   SSS             *
*                                                                *
*    TTTTT TTTTT III    DDDD  RRRR  III V   V EEEEE RRRR         *
*      T     T    I      D  D R   R  I  V   V E     R   R        *
*      T     T    I      D  D R   R  I  V   V E     R   R        *
*      T     T    I      D  D RRRR   I   V V  EEEE  RRRR         *
*      T     T    I      D  D R R    I   V V  E     R R          *
*      T     T    I      D  D R  R   I    V   E     R  R         *
*      T     T   III    DDDD  R   R III   V   EEEEE R   R        *
*                                                                *
*******************************************************************
*   Eyring Research Inst.  Copyright 1983,1987.
*   ALL RIGHTS RESERVED.
*
*        Module Name: TTI
*             Author: Richard Adams
*   Revision History:
*
*   10/03/86 3.0   Initial release
*   10/02/87 3.3   Update to 3.3 DRID
*
TTI IDNT 3.3  68K PDOS TTI DRIVER
*
******************************************************************
*
* This driver is intended  to input files from the terminal.   It
* gets characters from  the input  port  (PRT$)  of the task that
* opened it,stores them  in the buffer  (A3)  and  echoes them to
* active output port(s). It supports both XRLF read line and XRBF
* read block primitives.   OPEN call simply makes sure that there
* is an input port assigned to the task.  Close does nothing. EOF
* errors are returned, along with the byte count, if an escape
* is entered.
*
*   D5.L = Character count (-1 = Line)
*   D7.W = Channel status
*   (A2) = Driver base + 4
*   (A3) = Memory buffer
*   (A4) = File slot
*   (A5) = SYSRAM
*   (A6) = Task TCB
*   (A7) = Return address
*
    OPT  PDOS
```

```
*
  SECTION     0
DTTI      DC.W      $A55A+1   ;DRIVER ID
DROP      BRA.S     OPEN      ; 2 OPEN
DRCL      BRA.S     CLOS      ; 4 CLOSE
DRRD      BRA.S     READ      ; 6 READ
DRWR      BRA.S     WRIT      ; 8 WRITE
DRPS      MOVEQ.L   #70,D0    ;10 POSITION ERROR
          RTS
*
WRIT      MOVEQ.L   #80,D0    ;ERROR 80, DRIVER ERROR
          RTS
*
OPEN      TST.B     PRT$(A6)  ;IS THERE INPUT PORT?
          BEQ.S     WRIT      ;N, SEND ERROR 80
*
CLOS      CLR.W     D0        ;RETURN .EQ.
          RTS
*
*****************************************
*   READ CHARACTERS, BLOCK OR LINE
*
READ      MOVEQ.L   #0,D1     ;GET COUNT, EOF FOR ECSAPE
*
*   DO LINE/BLOCK READ
*
LINE      XGCR                ;GET A CHARACTER
          BLT.S     ESC       ;ESCAPE OUT
          TST.L     D5        ;LINE?
          BPL.S     @010      ;N, SKIP [CR] CHECK
          CMPI.B    #13,D0    ;Y, CR?
          BNE.S     @010      ;N, ECHO AND STORE
          CLR.B     (A3)      ;Y, TERMINATE LINE
          BRA  CLOS           ;GET BAT OUT
*
@010      XPCC                ;ECHO TO SCREEN
          MOVE.B    D0,(A3)+  ;SAVE IN BUFFER
          ADDQ.L    #1,D1     ;UP COUNT
          TST.L     D5        ;LINE?
          BMI.S     LINE      ;Y, SKIP COUNT CHECK
          CMP.L     D5,D1     ;N, DONE BLOCK COUNT?
          BLT.S     LINE      ;N, GET ANOTHER
          BRA.S     CLOS      ;Y, RETURN .EQ.
*
ESC       MOVE.L    D1,3*4+4(A7)   ;RETURN COUNT IN OLD D3
          MOVEQ.L   #56,D0         ;EOF ERROR RETURN
          RTS
*
*****************************************
*   DRIVER LENGTH CHECK
*
    IFLT 256-(*-DTTI+4)
    FAIL ** DRIVER LENGTH ERROR! **
    ENDC
*
    END  DTTI
```

## C.5 SSY68K/SIO-1/SIO-2 Driver Example

The following program is an example of a loadable driver for the SIO-1 and SIO-2 boards.

```
*   USSIO1:SR 07-APR-88
*******************************************************************
*                                                                 *
*  FFFFF  OOO  RRRR   CCC  EEEEE   U   U   A   RRRR  TTTTT  SSS   *
*  F     O   O R   R C   C E       U   U  A A  R   R   T   S   S  *
*  F     O   O R   R C     E       U   U A   A R   R   T   S      *
*  FFFF  O   O RRRR  C     EEEE    U   U AAAAA RRRR    T    SSS   *
*  F     O   O R R   C     E       U   U A   A R R     T       S  *
*  F     O   O R  R  C   C E       U   U A   A R  R    T   S   S  *
*  F      OOO  R   R  CCC  EEEEE    UUU  A   A R   R   T    SSS   *
*                                                                 *
*                   SSS   II   OOO     111                         *
*                  S   S  II  O   O   1111                         *
*                  S      II  O   O  11 11                         *
*                   SSS   II  O   O     11                         *
*                      S  II  O   O     11                         *
*                  S   S  II  O   O     11                         *
*                   SSS   II   OOO      11                         *
*******************************************************************
*
USSIO1 IDNT  1.5  FORCE STANDARD UART MODULE (INSTALLABLE FOR SIO)
*
* 15-JAN-88  1.3  FILE EXCERPT FROM FBIOSU:SR AND CHANGE UART TO
*                 AN INSTALLABLE UART
* 02-FEB-89  1.4  ADDED BAUDRATE 38400
* 07-APR-88  1.5  CHANGE MPCC BAUDRATE TABLE
*
*******************************************************************
*
    OPT  ARS,ALT
*
    XDEF USIO1
*
    IFUDF B.VEC  :B.VEC EQU 0
    INCLUDE    FPARM:SR
*
    SECTION   14
    PAGE
*
K1BEGN  EQU     $FC        ;OFFSET FOR ADDRESS OF KERNEL ENTRY POINT
K1SVEC  EQU     $20
K2CHRI  EQU     $14
B.PTMSK EQU     $2700      ;PORT DISABLE INTERRUPT MASK
*
```

```
*************************************************
*   PDOS CHARACTER I/O ROUTINES
*************************************************
*
* INSTALLABLE DRIVERS:
*
* EACH UART ENTRY IS DEFINED AS FOLLOWS:
*
*       UBEG  BRA.S     UDG            ;GET CHARACTER
*             BRA.S     UDP            ;PUT CHARACTER
*             BRA.S     UDB            ;BAUD UART
*             BRA.S     UDR            ;RESET UART
*             BRA.S     UDS            ;READ UART STATUS
*             BRA.S     UHW            ;HIGH WATER
*             BRA.S     ULW            ;LOW WATER
*             DC.B      'Ux'           ;UART ID
*             BRA.S     UDI            ;INSTALL DRIVER
*      UNAME  DS.B      'NAME',0       ;NAME OF DRIVER
*                                       ;(ZERO TERMINATED) EVEN
*             DC.W      $A557          ;IDENTIFIER
*             DC.W      P_TYP          ;PROCESSOR TYPE
*             BRA.W     UNINS          ;UNINSTALL
*
*   UARTS:     0(A2) = GET A CHARACTER  OUT: A0=BASE, D0=CHAR
*              2(A2) = PUT A CHARACTER  IN: A0=BASE, D0=CHAR,SR=^S^Q
*              4(A2) = BAUD THE PORT     IN: A0=BASE, D0=BAUDRATE
*              6(A2) = RESET THE PORT    IN: A0=BASE
*              8(A2) = READ PORT STATUS  IN: A0=BASE
*             10(A2) = HIGH WATER        IN: A0=BASE, D1=FLAGS
*             12(A2) = LOW WATER         IN: A0=BASE, D1=FLAGS
*
*               P_TYP  = %000000000000xxxx
*                                      \\\\__ 68000
*                                       \\\__ 68010
*                                        \\__ 68020
*                                         \__ 68030
*
*        F8BT. = FHPI 8DCS
BCSQ EQU 0    ;   \\\\ \\\\_________ 0 = ^S^Q ENABLE
BISC EQU 1    ;    \\\\ \\\_________ 1 = IGNORE CONTROL CHARACTER
BDTR EQU 2    ;     \\\\ \\_________ 2 = DTR ENABLE
B8CH EQU 3    ;      \\\\ \_________ 3 = 8 BIT CHARACTER ENABLE
BRIN EQU 4    ;       \\\\__________ 4 = RECEIVER INTERRUPTS ENABLE
BEVP EQU 5    ;        \\\__________ 5 = EVEN PARITY ENABLE
BHLW EQU 6    ;         \\__________ 6 = HIGH/LOW WATER (Reserved)
BFSQ EQU 7    ;          \__________ 7 = ^S^Q FLAG BIT (Reserved)
 PAGE
```

```
*************************************************
* UART ENTRIES ARE DEFINED AS FOLLOWS:
*
*   UxDG - GET CHARACTER
*
*           OUT:   D0.B = CHARACTER
*                  A0.L = UART BASE ADDRESS
*                    SR = EQ....CHARACTER FOUND
*                         NE....NO CHARACTER FOUND
*                         CS....CHARACTER FOUND BUT IGNORE
*
*        NOTE: 1)  ALL UARTS OF THE SAME TYPE MUST BE CHECKED
*                  FOR A CHARACTER.
*              2)  PRESERVE & RESTORE ALL REGISTERS USED.
*
*   UxDP - PUT CHARACTER
*
*            IN:   D0.B = CHARACTER
*                  D1.B = PORT FLAG (xxPI 8DBS)
*                  A0.L = UART BASE ADDRESS
*           OUT:     SR = EQ....CHARACTER OUTPUT
*                         NE....NO CHARACTER OUTPUT
*
*        NOTE: PRESERVE & RESTORE ALL REGISTERS.
*
*   UxDB - BAUD UART
*
*            IN:   D0.W = BAUD RATE (0-7)
*                  D1.B = PORT FLAG (xxPI 8DBS)
*                  A0.L = UART BASE ADDRESS
*           OUT:     SR = EQ....UART SUCCESSFULLY BAUDED
*                         NE....UART NOT SUCCESSFULLY BAUDED
*
*        NOTE: PRESERVE & RESTORE ALL REGISTERS.
*
*   UxDR - RESET UART
*
*            IN:   A0.L = UART BASE ADDRESS
*           OUT:     SR = EQ....UART SUCCESSFULLY RESET
*                         NE....UART NOT SUCCESSFULLY RESET
*
*        NOTE: PRESERVE & RESTORE ALL REGISTERS.
*
*   UxDS - READ UART STATUS
*
*            IN:   A0.L = UART BASE ADDRESS
*           OUT:   D0.W = UART STATUS
*
*        NOTE: PRESERVE & RESTORE ALL REGISTERS.
```

```
*
*       UDI   - INSTALL DRIVER
*                   IN: A1 = K1$BEGN
*                       A2 = OPTIONAL CARD BASE ADDRESS OR ZERO
*                       A5 = SYRAM BASE
*                       A6 = BEGIN OF TCB
*                     (A7) = RETURN ADDRESS
*                    4(A7) = RAM ADDRESS IN DSRTAB
*                  OUT: D0 = -1 ERROR
*                            NUMBER OF CARDS
*
*       UNINS - UNINSTALL DRIVER
*
*                   IN  (A7)  = RETURN ADDRESS
*                       4(A7) = RAM ADDRESS IN DSRTAB
*
    PAGE
*************************************************
*   UART TYPE #2:
*     SYS68K/SIO-1 BOARDS ON VME
*
*************************************************
 XDEF U$SIO1
*
SIO EQU  SIOBASE
PRINT '-->SYS68K/SIO-1/2  BOARD AS TYPE ',U.S4TYP,', ADR=$',$SIO,'
INCLUDED'
*
U.S4ADR  EQU  SIO+$000  ;SIO port #1   (MPCC 1)
U.S4TYP  EQU  2
*
U.S5ADR  EQU  SIO+$040  ;SIO port #2   (MPCC 2)
U.S5TYP  EQU  2
*
U.S6ADR  EQU  SIO+$080  ;SIO port #3   (MPCC 3)
U.S6TYP  EQU  2
*
U.S7ADR  EQU  SIO+$0C0  ;SIO port #4   (MPCC 4)
U.S7TYP  EQU  2
*
U.S8ADR  EQU  SIO+$100  ;SIO port #5   (MPCC 5)
U.S8TYP  EQU  2
*
U.S9ADR  EQU  SIO+$140  ;SIO port #6   (MPCC 6)
U.S9TYP  EQU  2
*
```

```
************************************************
 SIO2     EQU  SIOBASE+$200
 *
 U.SAADR  EQU  SIO2+$000 ;SIO #2 port #1     (MPCC 1)
 U.SATYP  EQU  2
 *
 U.SBADR  EQU  SIO2+$080 ;SIO #2 port #2     (MPCC 3)
 U.SBTYP  EQU  2
 *
 U.SCADR  EQU  SIO2+$100 ;SIO #2 port #3     (MPCC 5)
 U.SCTYP  EQU  2
 *
 U.SDADR  EQU  SIO2+$040 ;SIO #2 port #4     (MPCC 2)
 U.SDTYP  EQU  2
 *
 U.SEADR  EQU  SIO2+$0C0 ;SIO #2 port #5     (MPCC 4)
 U.SETYP  EQU  2
 *
 U.SFADR  EQU  SIO2+$140 ;SIO #2 port #6     (MPCC 6)
 U.SFTYP  EQU  2
 *
 ************************************************

*
 USIO1    BRA.S     U2DG      ;GET A CHARACTER    A0=BASE   D0=CHAR
          BRA.S     U2DP      ;PUT A CHARACTER    A0=BASE   D0=CHAR
          BRA.S     U2DBX     ;BAUD THE PORT      A0=BASE  D0=S/BAUDRATE
          BRA.S     U2DR      ;RESET THE PORT     A0=BASE
          BRA.S     U2DS      ;READ PORT STATUS   A0=BASE   D0=STATUS
          BRA.S     U2HW      ;HIGH WATER
          BRA.S     U2LW      ;LOW WATER
          DC.B      'U0'
          BRA.S     UDI
 UNAME    DC.B      'FORCE SIO-1/2',0
          EVEN
          DC.W      $A557
          DC.W      $F
          BRA.W     UNINS
          EVEN
 *
 UDI      BRA.W     INSTALL
 *
 U2DBX    BRA.W     U2DB            ;BRA.S TO SHORT
 U2HW     BRA.W     UHW             ;BRA.S TO SHORT
 U2LW     BRA.W     ULW             ;BRA.S TO SHORT
 ************************************************
 *   READ PORT STATUS
 *
 U2DS     MOVE.B    SSISR(A0),D0   ;READ STATUS BYTE
          RTS                      ;RETURN
 *
 ************************************************
 *   RESET UART
 *
 U2DR     BSR.S     U2DC           ;CHECK BASE AND RESET UART
          BRA.S     U2R_EQ         ;GOOD RETURN
 *
```

```
************************************************
 *   PUT CHARACTER

*
 U2DP     BTST.L    #BDTR,D1        ;CHECK DTR?
          BEQ.S     U2DP2           ;N
          BTST.B    #4,SSISR(A0)    ;Y, DTR (CTS),CHECK PIN 20,CTS
           STOPS XMIT
          BEQ.S     U2R_NE          ;NOT CLEAR TO SEND
 *
 U2DP2    TST.B     STSR(A0)        ;Y, CAN WE OUTPUT A CHAR TO FIFO?
          BPL.S     U2R_NE          ;N
          MOVE.B    D0,STDR(A0)     ;Y, OUTPUT IT
          BRA.S     U2R_EQ          ;RETURN .EQ.
 *
 ************************************************
 *      GET CHARACTER:
 *     We need this code for non-interrupt
 *     environments, like BOOT ROMs, etc.
 *
 U2DGRL   REG  D1/A1
 *
 U2DG     MOVEM.L   U2DGRL,-(A7)    ;SAVE REGS
          LEA.L     U$SIO1(PC),A1   ;POINT TO BASES
          MOVEQ.L   #0,D1           ;GET A 0
          MOVE.B    P$SIOF,D1       ;GET 0,1,2
          MULU.W    #6,D1           ;GET 0,6,12
          SUBQ.W    #1,D1           ;-1,5,11. ANY?
          BMI.S     U2DG04          ;N
 *
 U2DG2    MOVEA.L   (A1)+,A0        ;POINT TO MPCC BASE
          TST.B     SRSR(A0)        ;DATA AVAILABLE?
          DBMI      D1,U2DG02       ;N, LOOP 6 OR 12 TIMES
          BPL.S     U2DG04          ;LOOPED OUT, NO CHAR THERE!
          MOVE.B    SRDR(A0),D0     ;Y, GET CHARACTER
          MOVEM.L   (A7)+,U2DGRL    ;RESTORE A1
 *
 U2R_EQ   CMP.B     D0,D0           ;RETURN .EQ.
          RTS
 *
 U2DG4    MOVEM.L   (A7)+,U2DGRL    ;RESTORE A1
 *
 U2R_NE   CLR.W     -(A7)           ;SET STATUS .NE.
          RTR
 *
 ************************************************
 *   CHECK FOR VALID BASE ADDRESS
 *     & SET VECTOR # IF VALID
 *
 U2DCRL   REG  D1-D2/A1
 *
 U2DC     MOVEM.L   U2DCRL,-(A7)    ;SAVE REGS
          LEA.L     U$SIO1(PC),A1   ;POINT TO BASES
          MOVEQ.L   #63,D2          ;GET A VECTOR # COUNTER
          MOVEQ.L   #0,D1           ;GET A 0
          MOVE.B    P$SIOF,D1       ;GET 0,1,2
          MULU.W    #6,D1           ;GET 0,6,12
          SUBQ.W    #1,D1           ;-1,5,11. ANY?
          BMI.S     U2DC04          ;N
```

```
*
U2DC2    ADDQ.W    #1,D2           ;UP VECTOR # 64-75
         CMPA.L    (A1)+,A0        ;Y, VALID?
         DBEQ      D1,U2DC02       ;N, LOOP 6 OR 12 TIMES
         BNE.S     U2DC04          ;N, RETURN .NE.
         MOVE.W    #0,SPSR2-1(A0) ;Y, IF IN WORD MODE, PUT BACK INTO
                                    ;BYTE
         CLR.B     SECR(A0)        ;Y, CLEAR ERROR CNTRL
         MOVE.B    #$01,SRCR(A0)  ;SET RRES
         MOVE.B    #$01,STCR(A0)  ;SET TRES
         MOVE.B    D2,SRIVNR(A0)  ;SET VECTOR # = 64 TO 75
         MOVEM.L   (A7)+,U2DCRL   ;RESTORE
         RTS                       ;RETURN
*
U2DC4    MOVEM.L   (A7)+,U2DCRL   ;RESTORE
         ADDQ.W    #4,A7           ;POP RETURN ADDRESS
         BRA       U2R_NE
*
*************************************************
*
*    BAUD PORT
*
U2DB     BSR.S     U2DC            ;CHECK FOR VALID BASE
         MOVE.L    D2,-(A7)        ;SAVE SCRATCH
         CLR.B     SPSR1(A0)       ;PSR1=0
         MOVEQ.L   #$56,D2         ;ASSUME 7 BIT
         BTST      #B8CH,D1        ;8 BIT?
         BEQ.S     @002            ;N
         ADDQ.B    #8,D2           ;Y, TURN ON 8 BIT BIT
*
@002     MOVE.B    D2,SPSR2(A0)   ;SET CHAR LEN, STOP BITS
         MOVE.W    D0,D2
         LSL.W     #2,D2           ;TO LONG WORD INDEX
         MOVE.L    U2BRTB(PC,D2.W),D2 ;GET ENCODED BAUD RATE CONTANT
         MOVE.B    D2,SBRDR1(A0)  ;LSB
         ROR.W     #8,D2
         MOVE.B    D2,SBRDR2(A0)  ;MSB
         SWAP      D2              ;GET PRESCALER DIVIDER
         ANDI.B    #$0C,D2         ;ADD CLOCK SELECT TO PRESCALER
                                    ;DIVIDER
         MOVE.B    D2,SCCR(A0)    ;OUT CLOCK SELECT (DIV BY 2)
         CLR.W     D2              ;ASSUME NO PARITY
         BTST      #BEVP,D1        ;PARITY ENB?
         BEQ.S     @004            ;N
         MOVE.B    #$80,SECR(A0)  ;Y, ENB PARITY GEN/CHECK
*
@004     MOVE.B    #0,SECR(A0)    ;ENABLE/DISABLE PARITY
         CLR.B     STIER(A0)
         CLR.B     SSIER(A0)
         CLR.B     SRIER(A0)      ;NO RECV INTS
         BTST      #BRIN,D1        ;ENABLE INTS ?
         BNE.S     @006            ;N
         MOVE.B    #$80,SRIER(A0) ;Y, ENABLE INTS
*
*    This has been changed so that each SIO port uses vectors
*    #64 through #69, and #70 to #75 for SIO-1 ports:  RIVNR is
*    set in the U3DC routine to 64 to 75, so we ignore it.
*
```

```
@006      MOVE.B    #$C0,SSICR(A0)      ;SET RTS ON, DTR ASSERT
          CLR.B     SRCR(A0)            ;RESET RRES, ENABLE RECV
          MOVE.B    #$80,STCR(A0)       ;RESET TRES, ENABEL XMITER
          MOVE.L    (A7)+,D2            ;RESTORE
          BRA       U2R_EQ              ;GOOD RETURN (SR=EQ)
 *
 *
 *   MPCC BAUD TABLE
 *
 U2BRTB   DC.L      $100046             ;19200
          DC.L      $10008C             ; 9600
          DC.L      $100118             ; 4800
          DC.L      $000348             ; 2400
          DC.L      $100460             ; 1200
          DC.L      $000D20             ;  600
          DC.L      $001A40             ;  300
          DC.L      $102FBA             ;  110
          DC.L      $100023             ;38400
     EVEN
 *
 *****************************************
 UHW      MOVE.L    D0,-(A7)            ;SAVE D0
          MOVE.W    #$8000+'S'-'@',D0   ;GET NEGATE DTR AND ^S
          BRA.S     W2OUT
 *
 ULW      MOVE.L    D0,-(A7)
          MOVE.W    #$C000+'Q'-'@',D0   ;GET ASSERT DTR AND ^Q
 *
 W2OUT    BTST.L    #BDTR,D1            ;DTR?
          BEQ.S     @010                ;N, CHECK ^S^Q
          ROR.W     #8,D0               ;GET CONTROL BYTE
          MOVE.B    D0,SSICR(A0)        ;OUT DTR LEVEL
          BRA.S     @020                ;EXIT
 *
 @010     BTST.L    #BCSQ,D1            ;^S^Q?
          BEQ.S     @020                ;N, EXIT
          BSR.W     U2DP                ;Y, SEND CHARACTER, OK?
          BNE.S     @010                ;N, TRY AGAIN
 *
 @020          MOVE.L    (A7)+,D0            ;Y, RESTORE D0
          RTS
 *
 ****************************
 *       INSTALL
 *       DRIVER
 *
 INSTALL MOVEM.L A0,-(A7)
         TST.B   P$SIOF                 ;BOARD PRESENT ???
         BNE.S   @001
         MOVEM.L (A7)+,A0
         MOVE.L  #-1,D0
         RTS
 *
 * SIO-1/-2 VECTOR DEFINITIONS
 *
```

```
@001     MOVE.W   #64,D0         ;PORT 4 SERVICE (SIO-1 #1)
         LEA.L    SIO4(PC),A0
         JSR      K1SVEC(A1)     ;LOAD NEW INT VECTOR
         ADDQ.W   #1,D0          ;PORT 5 SERVICE
         LEA.L    SIO5(PC),A0
         JSR      K1SVEC(A1      ;LOAD NEW INT VECTOR
         ADDQ.W   #1,D0          ;PORT 6 SERVICE
         LEA.L    SIO6(PC),A0
         JSR      K1SVEC(A1)     ;LOAD NEW INT VECTOR
         ADDQ.W   #1,D0          ;PORT 7 SERVICE
         LEA.L    SIO7(PC),A0
         JSR      K1SVEC(A1)     ;LOAD NEW INT VECTOR
         ADDQ.W   #1,D0          ;PORT 8 SERVICE
         LEA.L    SIO8(PC),A0
              JSR      K1SVEC(A1)     ;LOAD NEW INT VECTOR
         ADDQ.W   #1,D0          ;PORT 9 SERVICE
         LEA.L    SIO9(PC),A0
         JSR      K1SVEC(A1)     ;LOAD NEW INT VECTOR
*
         ADDQ.W   #1,D0          ;PORT A SERVICE (SIO-1 #2)
         LEA.L    SIOA(PC),A0
         JSR      K1SVEC(A1)     ;LOAD NEW INT VECTOR
         ADDQ.W   #1,D0          ;PORT B SERVICE
         LEA.L    SIOB(PC),A0
         JSR      K1SVEC(A1)     ;LOAD NEW INT VECTOR
         ADDQ.W   #1,D0          ;PORT C SERVICE
         LEA.L    SIOC(PC),A0
         JSR      K1SVEC(A1)     ;LOAD NEW INT VECTOR
         ADDQ.W   #1,D0          ;PORT D SERVICE
         LEA.L    SIOD(PC),A0
         JSR      K1SVEC(A1)     ;LOAD NEW INT VECTOR
         ADDQ.W   #1,D0          ;PORT E SERVICE
         LEA.L    SIOE(PC),A0
         JSR      K1SVEC(A1)     ;LOAD NEW INT VECTOR
         ADDQ.W   #1,D0          ;PORT F SERVICE
         LEA.L    SIOF(PC),A0
         JSR      K1SVEC(A1)     ;LOAD NEW INT VECTOR
         MOVEM.L  (A7)+,A0
         CLR.L    D0
         MOVE.B   P$SIOF,D0      ;RETURN NUMBER OF CARDS FOUND
         RTS
*
****************************
*        UNINSTALL
*        DRIVER
*
UNINS    MOVEM.L D0/A0,-(A7)
         MOVE.W  #64,D0          ;UNINSTALL
@001     MOVEA.L $3FC,A0         ;LOAD DEFAULT INTERRUPT VECTOR
         XVEC
         ADDQ.W  #1,D0           ;(# 64 -75 USED BY SIO1/2)
         CMP.W   #76,D0
         BLT.S   @001
         MOVEM.L (A7)+,D0/A0
         RTS
```

```
*
U$SIO1   DC.L U.S4ADR,U.S5ADR,U.S6ADR,U.S7ADR    ;SIO #1 port #1-4
         DC.L U.S8ADR,U.S9ADR                    ;SIO #1 port #5-6
         DC.L U.SAADR,U.SBADR,U.SCADR,U.SDADR    ;SIO #2 port #1-4
         DC.L U.SEADR,U.SFADR                    ;SIO #2 port #5-6
*
*******************************************************************
*
*   INT HANDLER FOR TYPE 2 UARTS --- SIO-1/SIO-2
*
SIO4     MOVE.W    #0*4,-(A7)     ;GET SIO #1 PORT1 ADDR
         BRA.S     SIOPINT
SIO5     MOVE.W    #1*4,-(A7)     ;GET SIO #1 PORT2 ADDR
         BRA.S     SIOPINT
SIO6     MOVE.W    #2*4,-(A7)     ;GET SIO #1 PORT3 ADDR
         BRA.S     SIOPINT
SIO7     MOVE.W    #3*4,-(A7)     ;GET SIO #1 PORT4 ADDR
         BRA.S     SIOPINT
SIO8     MOVE.W    #4*4,-(A7)     ;GET SIO #1 PORT5 ADDR
         BRA.S     SIOPINT
SIO9     MOVE.W    #5*4,-(A7)     ;GET SIO #1 PORT6 ADDR
         BRA.S     SIOPINT
SIOA     MOVE.W    #6*4,-(A7)     ;GET SIO #2 PORT1 ADDR
         BRA.S     SIOPINT
SIOB     MOVE.W    #7*4,-(A7)     ;GET SIO #2 PORT2 ADDR
         BRA.S     SIOPINT
SIOC     MOVE.W    #8*4,-(A7)     ;GET SIO #2 PORT3 ADDR
         BRA.S     SIOPINT
SIOD     MOVE.W    #9*4,-(A7)     ;GET SIO #2 PORT4 ADDR
         BRA.S     SIOPINT
SIOE     MOVE.W    #10*4,-(A7)    ;GET SIO #2 PORT5 ADDR
         BRA.S     SIOPINT
SIOF     MOVE.W    #11*4,-(A7)    ;GET SIO #2 PORT6 ADDR
*
SIOPINT  MOVE.W    #B.PTMSK,SR    ;DISABLE INTS
         MOVE.W    (A7)+,P$UADR   ;POP TABLE INDEX
         MOVEM.L   D0-A6,-(A7)    ;SAVE REGS
         LEA.L     U$SIO1(PC),A0  ;POINT TO TABLE
         ADDA.W    P$UADR,A0      ;ADD BASE ADDR OFFSET
         MOVEA.L   (A0),A0        ;GET BASE ADDRESS
         TST.B     SRSR(A0)       ;DATA AVAILABLE?
         BPL.S     @04            ;N, ?????
         MOVE.B    SRDR(A0),D0    ;Y, GET CHARACTER
*
@02      MOVEA.L   B$SRAM,A5      ;LOAD UP SYSRAM PTR
         MOVEA.L   K1BEGN(A5),A1  ;LOAD KERNEL ENTRY POINT
         JMP       K2CHRI(A1)     ;TO KERNEL K2$CHRI
*
@04      MOVEM.L   (A7)+,D0-A6    ;RESTORE REGS
         RTE                      ;RETURN & HOPE
*
*************************************************
*
        END  USIO1
```

## C.6 SYS68K/ISIO-1/ISIO-2 Driver Example

The following program is an example of a loadable driver for the ISIO-1 and ISIO-2 boards.

```
*       USISIO1:SR      02-MAR-88
*******************************************************************
*                                                                 *
*   FFFFF  OOO  RRRR   CCC  EEEEE   U   U   A   RRRR  TTTTT  SSS  *
*   F     O   O R   R C   C E       U   U  A A  R   R   T   S   S *
*   F     O   O R   R C     E       U   U A   A R   R   T   S     *
*   FFFF  O   O RRRR  C     EEEE    U   U AAAAA RRRR    T    SSS  *
*   F     O   O R R   C     E       U   U A   A R R     T       S *
*   F     O   O R  R  C   C E       U   U A   A R  R    T   S   S *
*   F      OOO  R   R  CCC  EEEEE    UUU  A   A R   R   T    SSS  *
*                                                                 *
*                II   SSS   II   OOO     111                      *
*                II  S   S  II  O   O   1111                      *
*                II  S      II  O   O  11 11                      *
*                II   SSS   II  O   O     11                      *
*                II      S  II  O   O     11                      *
*                II  S   S  II  O   O     11                      *
*                II   SSS   II   OOO      11                      *
*                                                                 *
*******************************************************************
*
USISIO1 IDNT  1.4  FORCE STANDARD UART MODULE (INSTALLABLE ISIO)
*
*  15-JAN-88  1.3  FILE EXCERPT FROM FBIOSU:SR AND CHANGE UART TO
*                  AN INSTALLABLE UART
*  02-MAR-88  1.4  FIXED FOR BAUDRATE 38400 NOT SUPPORTED
*
*******************************************************************
*
        OPT     ARS,ALT
*
        XDEF    UISIO1
*
        IFUDF B.VEC  :B.VEC EQU 0
        INCLUDE  FPARM:SR
*
        SECTION 14
        PAGE
*
K1BEGN  EQU     $FC            ;OFFSET FOR ADDRES OF KERNEL ENTRY POINT
K1SVEC  EQU     $20
K2CHRI  EQU     $14
B.PTMSK EQU     $2700           ;PORT DISABLE INTERRUPT MASK
*
```

```
*************************************************
*       PDOS CHARACTER I/O ROUTINES
*************************************************
*
* INSTALLABLE DRIVERS:
*
* EACH UART ENTRY IS DEFINED AS FOLLOWS:
*
*       UBEG    BRA.S   UDG             ;GET CHARACTER
*               BRA.S   UDP             ;PUT CHARACTER
*               BRA.S   UDB             ;BAUD UART
*               BRA.S   UDR             ;RESET UART
*               BRA.S   UDS             ;READ UART STATUS
*               BRA.S   UHW             ;HIGH WATER
*               BRA.S   ULW             ;LOW WATER
*               DC.B    'Ux'            ;UART ID
*               BRA.S   UDI             ;INSTALL DRIVER
*       UNAME   DS.B    'NAME',0        ;NAME OF DRIVER (ZERO
*                                       ;TERMINATED)
*               EVEN
*               DC.W    $A557           ;IDENTIFIER
*               DC.W    P_TYP           ;PROCESSOR TYPE
*               BRA.W   UNINS           ;UNINSTALL
*
*      UARTS:   0(A2) = GET A CHARACTER  OUT: A0=BASE, D0=CHAR
*               2(A2) = PUT A CHARACTER   IN: A0=BASE, D0=CHAR, SR=^S^Q
*               4(A2) = BAUD THE PORT     IN: A0=BASE, D0=BAUDRATE
*               6(A2) = RESET THE PORT    IN: A0=BASE
*               8(A2) = READ PORT STATUS  IN: A0=BASE
*              10(A2) = HIGH WATER        IN: A0=BASE, D1=FLAGS
*              12(A2) = LOW WATER         IN: A0=BASE, D1=FLAGS
*
*              P_TYP  = %000000000000xxxx
*                                    \\\\__ 68000
*                                     \\\__ 68010
*                                      \\__ 68020
*                                       \__ 68030
*
*               F8BT. = FHPI 8DCS
BCSQ    EQU     0       ;\\\\ \\\\____ 0 = ^S^Q ENABLE
BISC    EQU     1       ; \\\\ \\\____ 1 = IGNORE CONTROL CHARACTER
BDTR    EQU     2       ;  \\\\ \\____ 2 = DTR ENABLE
B8CH    EQU     3       ;   \\\\ \____ 3 = 8 BIT CHARACTER ENABLE
BRIN    EQU     4       ;    \\\\_____ 4 = RECEIVER INTERRUPTS ENABLE
BEVP    EQU     5       ;     \\\_____ 5 = EVEN PARITY ENABLE
BHLW    EQU     6       ;      \\_____ 6 = HIGH/LOW WATER (Reserved)
BFSQ    EQU     7       ;       \_____ 7 = ^S^Q FLAG BIT (Reserved)
*
```

```
*************************************************
* UART ENTRIES ARE DEFINED AS FOLLOWS:
*
*       UxDG - GET CHARACTER
*
*              OUT: D0.B = CHARACTER
*                   A0.L = UART BASE ADDRESS
*                     SR = EQ....CHARACTER FOUND
*                          NE....NO CHARACTER FOUND
*                          CS....CHARACTER FOUND BUT IGNORE
*
*               NOTE: 1) ALL UARTS OF THE SAME TYPE MUST BE CHECKED
*                        FOR A CHARACTER.
*                     2) PRESERVE & RESTORE ALL REGISTERS USED.
*
*       UxDP - PUT CHARACTER
*
*                   IN: D0.B = CHARACTER
*                       D1.B = PORT FLAG (xxPI 8DBS)
*                       A0.L = UART BASE ADDRESS
*                  OUT:   SR = EQ....CHARACTER OUTPUT
*                              NE....NO CHARACTER OUTPUT
*
*               NOTE: PRESERVE & RESTORE ALL REGISTERS.
*
*       UxDB - BAUD UART
*
*                   IN: D0.W = BAUD RATE (0-7)
*                       D1.B = PORT FLAG (xxPI 8DBS)
*                       A0.L = UART BASE ADDRESS
*                  OUT:   SR = EQ....UART SUCCESSFULLY BAUDED
*                              NE....UART NOT SUCCESSFULLY BAUDED
*
*               NOTE: PRESERVE & RESTORE ALL REGISTERS.
*
*       UxDR - RESET UART
*
*                   IN: A0.L = UART BASE ADDRESS
*                  OUT:   SR = EQ....UART SUCCESSFULLY RESET
*                              NE....UART NOT SUCCESSFULLY RESET
*
*               NOTE: PRESERVE & RESTORE ALL REGISTERS.
*
*       UxDS - READ UART STATUS
*
*                   IN: A0.L = UART BASE ADDRESS
*                  OUT: D0.W = UART STATUS
*
*               NOTE: PRESERVE & RESTORE ALL REGISTERS.
*
*       UDI   - INSTALL DRIVER
*                   IN: A1 = K1$BEGN
*                       A2 = OPTIONAL CARD BASE ADDRESS OR ZERO
*                       A5 = SYRAM BASE
*                       A6 = BEGIN OF TCB
*                     (A7) = RETURN ADDRESS
*                    4(A7) = RAM ADDRESS IN DSRTAB
*                  OUT: D0 = -1 ERROR
*                            NUMBER OF CARDS
*
```

```
*       UNINS - UNINSTALL DRIVER
*
*                   IN  (A7)  = RETURN ADDRESS
*                      4(A7) = RAM ADDRESS IN DSRTAB
*
 *******************************************************************
*       UART TYPE #3:
*         SYS68K/ISIO-1/-2 BOARDS ON VME
*
*       ISIO-1/2 HANDLING ROUTINES
*
*               - PUTCHAR
*               - RESET PORT
*               - PORT STATUS
*               - BAUD PORT
*               - HIGH WATER
*               - LOW WATER
*
*
*
*       DEF ALL OTHER ISIO-1/2 ADDRESSES
*
        XDEF    U$ISIO                          ;TABLE ADDRESS
*
U.I4ADR EQU     ISIOB+$8000
U.I5ADR EQU     ISIOB+$8020
U.I6ADR EQU     ISIOB+$8040
U.I7ADR EQU     ISIOB+$8060
U.I8ADR EQU     ISIOB+$8080
U.I9ADR EQU     ISIOB+$80A0
U.IAADR EQU     ISIOB+$80C0
U.IBADR EQU     ISIOB+$80E0
U.ICADR EQU     ISIOB2+$8000
U.IDADR EQU     ISIOB2+$8020
U.IEADR EQU     ISIOB2+$8040
U.IFADR EQU     ISIOB2+$8060
U.IGADR EQU     ISIOB2+$8080
U.IHADR EQU     ISIOB2+$80A0
U.IIADR EQU     ISIOB2+$80C0
U.IJADR EQU     ISIOB2+$80E0
*
U.I4TYP EQU     3
U.I5TYP EQU     3
U.I6TYP EQU     3
U.I7TYP EQU     3
U.I8TYP EQU     3
U.I9TYP EQU     3
U.IATYP EQU     3
U.IBTYP EQU     3
U.ICTYP EQU     3
U.IDTYP EQU     3
U.IETYP EQU     3
U.IFTYP EQU     3
U.IGTYP EQU     3
U.IHTYP EQU     3
U.IITYP EQU     3
U.IJTYP EQU     3
*
*
PRINT '-->SYS68K/ISIO-1/2 BOARD AS TYPE ',U.I4TYP,', ADR=$',
```

```
$ISIOB,' INCLUDED'
UISIO1  BRA.S   U3DG            ;GET A CHARACTER   A0=BASE D0=CHAR
        BRA.S   U3DP            ;PUT A CHARACTER   A0=BASE D0=CHAR
        BRA.S   U3DBX           ;BAUD THE PORT     A0=BASE
D0=S/BAUDRATE
        BRA.S   U3DR            ;RESET THE PORT    A0=BASE
        BRA.S   U3DS            ;READ PORT STATUS  A0=BASE D0=STATUS
        BRA.S   U3HW            ;HIGH WATER
        BRA.S   U3LW            ;LOW WATER
        DC.B    'U0'            ;ID
        BRA.S   UDI             ;INSTALL
UNAME   DC.B    'FORCE ISIO-1/2',0
        EVEN
        DC.W    $A557
        DC.W    $F
        BRA.W   UNINS            ;UNINSTALL
*
UDI     BRA.W   INSTALL          ;BRA.S TO SHORT
U3LW    BRA.W   ULW              ;BRA.S TO SHORT
U3DBX   BRA.W   BAUDP            ;BRA.S TO SHORT
*
*
*       GET CHARACTER ALWAYS RETURNS WITH ERROR
*
U3DG    BRA     U3R_NE          ;RETURN .NE.
*
*       PUT A CHAR TO AN ISIO PORT
*
U3DP    BRA.L   PUTC
*
*       RESET THE PORT
*
U3DR    MOVEM.L D0-D2/A0-A1,-(A7);SAVE USED REGISTERS
        MOVE.L  A0,D0           ;GET PORT BASE
        LSR.L   #4,D0           ;GET ISIO TASK#
        ANDI.L  #$F,D0          ;MASK UNNECESSARY BITS
        ADDQ.L  #1,D0           ;EXTENDED TO CHANNEL#
@001    MOVE.L  A0,D1           ;LOAD PORT BASE
        ANDI.L  #$FFFF0000,D1   ;GET BOARD BASE
        MOVEA.L D1,A1           ;SAVE BOARD BASE
        MOVE.L  #ISABORT,D1     ;SET ABORT OFFSET
        MOVE.W  D0,0(A1,D1.L)   ;SET CHANNEL # TO ABORT
        MOVE.B  ISINT(A1),D2    ;EXECUTES A LOCAL ISIO INT
@002    TST.W   0(A1,D1.L)      ;WAIT UNTIL DONE
        BNE.S   @002
        MOVE.L  #$10000,D1      ;DELAY
@003    SUBQ.L  #1,D1
        BNE.S   @003
        TST.W   (A0)            ;CHANNEL READY ?
        BPL.S   @001            ;NO, DO AGAIN
        MOVEM.L (A7)+,D0-D2/A0-A1       ;RESTOR REGISTERS
        BRA     U3R_EQ          ;RETURN SUCCESSFUL
```

```
*
*       READ PORT STATUS
*
U3DS    CLR.W   D0                 ;ALWAYS RETURNS 0
        BRA     U3R_EQ
*
*       HIGH WATER
*
U3HW    MOVEM.L D0-D1,-(A7)
        MOVE.L  A0,D0
        ROR.W   #5,D0              ;GET PORT NUMBER ON BOARD
        AND.L   #$07,D0
        CMPA.L  #ISIOB2,A0
        BLT.S   @02
        OR.W    #$8,D0             ;SECOND BOARD BITS
@02     MOVE.W  P$ISINT,D1
        BSET    D0,D1              ;SET HIGH WATER FLAG OF CHANNEL
        MOVE.W  D1,P$ISINT
        MOVEM.L (A7)+,D0-D1        ;RESTORE REGISTERS
        BRA     U3R_EQ
*
*       LOW WATER
*
ULW     MOVEM.L D0-D1,-(A7)
        MOVE.L  A0,D0
        ROR.W   #5,D0              ;GET PORT NUMBER ON BOARD
        AND.L   #$07,D0
        CMP.L   #ISIOB2,A0
        BLT.S   @02
        OR.W    #$8,D0             ;SECOND BOARD BITS
@02     MOVE.W  P$ISINT,D1
        BCLR    D0,D1              ;CLEAR HIGHWATER FLAG OF CHANNEL
        MOVE.W  D1,P$ISINT
        MOVEM.L (A7)+,D0-D1        ;RESTORE REGISTERS
        MOVE.W  #GETCHI,(A0)       ;GET CHAR WITH INTERRUPT
        BRA     U3R_EQ
*
****************************************
*
*       BAUD PORT
*
BAUDP   TST.B   P$ISIOF               ;BOARD PRESENT ??
        BEQ     U3R_NE                ;N, RETURN ERROR
        CMP.L   #ISIOB2,A0            ;BOARD #2 ??
        BLT.S   @001                  ;N
        CMPI.B  #2,P$ISIOF            ;TWO BOARDS ??
        BLT     U3R_NE                ;N, ERROR
*
@001    MOVEM.L D0-D4/A0-A2,-(A7)  ;SAVE REGISTERS
        LEA.L   U$ISIO(PC),A1         ;GET PORT TABLE
        MOVEQ.L #16,D3                ;SET COUNTER
@003    CMPA.L  (A1)+,A0
        BEQ.S   @004                  ;Y, FOUND
        SUBQ.W  #1,D3                 ;N, DECREMENT COUNTER
        BNE.S   @003                  ;REPEAT
        MOVEM.L (A7)+,D0-D4/A0-A2  ;NOT FOUND, ERROR RETURN
        BRA     U3R_NE
```

```
*
 @004    ADDA.L  #$10,A0          ;POINT TO OUTPUT CHANNEL
 @010    TST.W   (A0)             ;WAIT UNTIL CHANNEL IS READY
         BPL.S   @010
         MOVE.L  A0,D4            ;GET CMDRAM
         ANDI.L  #$FF,D4          ;MASK UNNECESARY BITS
         LSL.L   #8,D4            ;GET BUFFER OFFSET
         MOVE.L  A0,D3            ;GET ADDRESS
         AND.L   #$FFFF8000,D3
         MOVE.L  D3,A1            ;GET REAL BASE
         ADDA.L  D4,A1            ;ADD OFFSET
         CLR.W   D3               ;CLEAR LOWER WORD OF ADDRESS
         MOVE.L  A1,4(A0)         ;STORE POINTER IN CMDRAM
         SUB.L   D3,4(A0)         ;MINUS BOARD BASE
         PAGE
 *
 *       INIT THE HANDSHAKE MODE
 *
         CLR.W   D4
         BTST    #BDTR,D1         ;TEST DTR BIT
         BEQ.S   @005
         BSET    #0,D4
 @005    BTST    #BCSQ,D1         ;TEST XON/XOFF BIT
         BEQ.S   @006
         BSET    #1,D4
 @006    MOVE.B  D4,(A1)+         ;SET MODE BYTE
 *
 *       HANDLE CHARACTER LENGTH
 *
 @007    BTST    #B8CH,D1         ;7 OR 8 BIT CHARS?
         BNE.S   @008             ;8
         MOVE.B  #$02,(A1)+       ;7, STORE IT
         BRA.S   @009
 @008    MOVE.B  #$03,(A1)+       ;8, STORE IT
 *
 *       NUMBER OF STOP BITS IS ALWAYS 1
 *
 @009    MOVE.B  #$07,(A1)+
 *
 *       CHECK PARITY FLAG
 *
         BTST    #BEVP,D1         ;EVEN OR NONE?
         BNE.S   @011             ;EVEN
         MOVE.B  #0,(A1)+         ;NO PARITY
         BRA.S   @013
 @011    MOVE.B  #$02,(A1)+       ;EVEN PARITY
 *
 *       GET BAUDRATE AND START INIT
 *
 @013    LEA.L   ISBTB(PC),A2      ;GET ISIO BAUDRATE TABLE
         MOVE.W  D0,D3             ;GET BAUDRATE
         ANDI.W  #%00000111,D3     ;MASK BAUDRATE (JUST IN CASE)
         MOVE.B  0(A2,D3.W),(A1)+ ;GET ENCODED BAUD RATE
         MOVE.B  0(A2,D3.W),(A1)+ ;TWICE FOR TX AND RX
         MOVE.W  #ASINI,(A0)       ;ASYNINI COMMAND
 @014    TST.W   (A0)              ;WAIT FOR DONE
         BPL.S   @014
```

```
*
*       CHECK INT DISABLE FLAG
*
@012    BTST    #BRIN,D1
        BEQ.S   @015
        BSR     U3DR                ;RESET PORT
        BRA.S   @016
        PAGE
*
*       INIT THE PORT FOR INTERRUPTED INPUT
*
@015    SUBA.L  #$10,A0             ;GET INPUT CHANNEL
        MOVE.W  #GETCHI,(A0)        ;GET CHAR WITH INTERRUPT
*
@016    MOVE.L  A0,D0
        ROR.W   #5,D0               ;GET PORT NUMBER ON BOARD
        AND.W   #$07,D0
        CMP.L   #ISIOB2,A0
        BLT.S   @018
        OR.W    #$8,D0              ;SECOND BOARD BITS
@018    MOVE.W  P$ISINT,D1
        BCLR    D0,D1               ;CLEAR HIGHWATER FLAG OF CHANNEL
        MOVE.W  D1,P$ISINT
        MOVEM.L (A7)+,D0-D4/A0-A2 ;RESTORE REGISTERS
*
*
U3R_EQ  MOVE.W  #$04,-(A7)          ;RETURN .EQ.
        RTR
*
*
U3R_NE  MOVE.W  #$0,-(A7)           ;RETURN .NE.
        RTR
```

```
*
*
PUTC     MOVE.L  A0,-(A7)          ;SAVE CHANNEL BASE
         ADDA.L  #$10,A0           ;GET OUTPUT CHANNEL
@000     TST.W   (A0)              ;CHANNEL READY?
         BPL.S   @000              ;N
         CLR.W   2(A0)
         MOVE.B  D0,3(A0)          ;Y, WRITE THE CHAR
         MOVE.W  #PUTCH,(A0)       ;WRITE ISIO COMMAND
@001     TST.W   (A0)              ;WAIT UNTIL DONE
         BPL.S   @001
         MOVE.L  (A7)+,A0          ;RESTORE PORT BASE
         BRA     U3R_EQ            ;RETURN SUCCESSFUL
*
****************************
*        INSTALL/UNINSTALL
*        DRIVER
*
RL       REG     D1-D1/A0-A3
*
INSTALL MOVEM.L RL,-(A7)
         TST.B   P$ISIOF           ;BOARD PRESENT ??
         BNE.S   @001              ;
         MOVEM.L (A7)+,RL          ;RESTORE REGS
         MOVE.L  #-1,D0            ;N, RETURN ERROR
         RTS
@001     MOVE.B  P$ISIOF,D1        ;NUMBER OF CARDS
         LEA.L   ISIOH1(PC),A3     ;GET INT SERVICE ADDR. FOR PORT 1
         LEA.L   ISIOB,A2          ;BASE ADDRESS
*
*****************************************************************
*        INIT THE ISIO ONBOARD BIM FOR INTERRUPT GENERATION
*    IN: A3 = INTERRUPT ROUTINE ADDRESS
*        A2 = CHANNEL ADDRESS
*        D1 = NUMBER OF CARDS   ( 1,2 cards)
*
INIBIM   MOVE.L  A2,D0             ;GET CHANNEL ADDRESS
         AND.L   #$FFFE0000,D0     ;MASK
         MOVE.L  D0,A2
         MOVE.B  #$54,ISBCR0(A2) ;SET BIM CONTROL REGS
         MOVE.B  #$54,ISBCR1(A2)
         MOVE.B  #$54,ISBCR2(A2)
         MOVE.B  #$54,ISBCR3(A2)
         CMP.B   #0,D1
         BEQ.S @010                ;IF BOARD 2 OTHER VECTORS
         MOVEA.L A3,A0             ;LOAD INTERRUPT ROUTINE ADDRESS
         MOVE.B  #76,D0            ;GET VECTOR NUMBER IN D0
         MOVE.B  D0,ISBVR0(A2)     ;N, SET BIM VECTOR REGS FOR BOARD #1
         JSR     K1SVEC(A1)        ;SET NEW BUS ERROR FOR ISIO
         ADDQ.W  #6,A3             ;GET NEW INTERRUPT ROUTINE ADDRESS
         MOVEA.L A3,A0             ;LOAD INTERRUPT ROUTINE ADDRESS
         MOVE.B  #77,D0            ;GET VECTOR NUMBER IN D0
         MOVE.B  D0,ISBVR1(A2)
         JSR     K1SVEC(A1)        ;SET NEW BUS ERROR FOR ISIO
         ADDQ.W  #6,A3             ;GET NEW INTERRUPT ROUTINE ADDRESS
         MOVEA.L A3,A0             ;LOAD INTERRUPT ROUTINE ADDRESS
         MOVE.B  #78,D0            ;GET VECTOR NUMBER IN D0
         MOVE.B  D0,ISBVR2(A2)
         JSR     K1SVEC(A1)        ;SET NEW BUS ERROR FOR ISIO
         ADDQ.W  #6,A3             ;GET NEW INTERRUPT ROUTINE ADDRESS
```

```
        MOVEA.L A3,A0           ;LOAD INTERRUPT ROUTINE ADDRESS
        MOVE.B  #79,D0          ;GET VECTOR NUMBER IN D0
        MOVE.B  D0,ISBVR3(A2)
        JSR     K1SVEC(A1)      ;SET NEW BUS ERROR FOR ISIO
        ADDQ.W  #6,A3           ;GET NEW INTERRUPT ROUTINE ADDRESS
        CMP.W   #2,D1
          BNE.S OUT             ;ONLY ONE BOARD
        CLR.W   D1
        ADDA.L  #20000,A2       ;LOAD NEXT BASE ADDRESS
          BRA.S INIBIM
@010    MOVEA.L A3,A0           ;LOAD INTERRUPT ROUTINE ADDRESS
        MOVE.B  #80,D0          ;GET VECTOR NUMBER IN D0
        MOVE.B  D0,ISBVR0(A2)   ;SET BIM VECTOR REGS FOR BOARD #2
        JSR     $20(A1)         ;
        ADDQ.W  #6,A3           ;GET NEW INTERRUPT ROUTINE ADDRESS
        MOVEA.L A3,A0           ;LOAD INTERRUPT ROUTINE ADDRESS
        MOVE.B  #81,D0          ;GET VECTOR NUMBER IN D0
        MOVE.B  D0,ISBVR1(A2)
        JSR     $20(A1)         ;
        ADDQ.W  #6,A3           ;GET NEW INTERRUPT ROUTINE ADDRESS
        MOVEA.L A3,A0           ;LOAD INTERRUPT ROUTINE ADDRESS
        MOVE.B  #82,D0          ;GET VECTOR NUMBER IN D0
        MOVE.B  D0,ISBVR2(A2)
        JSR     $20(A1)         ;
        ADDQ.W  #6,A3           ;GET NEW INTERRUPT ROUTINE ADDRESS
        MOVEA.L A3,A0           ;LOAD INTERRUPT ROUTINE ADDRESS
        MOVE.B  #83,D0          ;GET VECTOR NUMBER IN D0
        MOVE.B  D0,ISBVR3(A2)
        JSR     $20(A1)         ;
OUT     MOVEM.L (A7)+,RL        ;RESTORE REGS
        CLR.L   D0
        MOVE.B  P$ISIOF,D0      ;RETURN # OF CARDS FOUND
        RTS
*
****************************
*       UNINSTALL
*       DRIVER
*
UNINS   MOVEM.L D0/A0,-(A7)
        MOVE.W  #76,D0          ;UNINSTALL
@001    MOVEA.L $3FC,A0         ;LOAD DEFAULT INTERRUPT VECTOR
        XVEC
        ADDQ.W  #1,D0           ;(# 76 - 83 USED BY ISIO1/2)
        CMP.W   #84,D0
        BLT.S   @001
        MOVEM.L (A7)+,D0/A0
        RTS
```

```
*
*       ISIO BAUD RATE TABLE
*
ISBTB   DC.B    $0E             ;19.200
        DC.B    $0D             ; 9.600
        DC.B    $0C             ; 4.800
        DC.B    $0B             ; 2.400
        DC.B    $09             ; 1.200
        DC.B    $07             ;   600
        DC.B    $06             ;   300
        DC.B    $02             ;   110
        DC.B    $0E             ;19.200 (38.400 NOT SUPPORTED)
        EVEN
        PAGE
*
U$ISIO  DC.L    U.I4ADR
        DC.L    U.I5ADR
        DC.L    U.I6ADR
        DC.L    U.I7ADR
        DC.L    U.I8ADR
        DC.L    U.I9ADR
        DC.L    U.IAADR
        DC.L    U.IBADR
        DC.L    U.ICADR
        DC.L    U.IDADR
        DC.L    U.IEADR
        DC.L    U.IFADR
        DC.L    U.IGADR
        DC.L    U.IHADR
        DC.L    U.IIADR
        DC.L    U.IJADR

****************************************************************
*
*  THE FOLLOWING CODE IS THE ISIO-1/2 CHARACTER PROCESSOR
*
ISIOH1  MOVE.W  #0,-(A7)        ;OFFSET TO CMD RAM
        BRA.S   ISIOHC
ISIOH3  MOVE.W  #$40,-(A7)      ;OFFSET TO CMD RAM
        BRA.S   ISIOHC
ISIOH5  MOVE.W  #$80,-(A7)      ;OFFSET TO CMD RAM
        BRA.S   ISIOHC
ISIOH7  MOVE.W  #$C0,-(A7)      ;OFFSET TO CMD RAM
        BRA.S   ISIOHC
ISIOH9  MOVE.W  #$100,-(A7)     ;OFFSET TO CMD RAM, BOARD #2
        BRA.S   ISIOHC
ISIOH11 MOVE.W  #$140,-(A7)     ;OFFSET TO CMD RAM
        BRA.S   ISIOHC
ISIOH13 MOVE.W  #$180,-(A7)     ;OFFSET TO CMD RAM
        BRA.S   ISIOHC
ISIOH15 MOVE.W  #$1C0,-(A7)     ;OFFSET TO CMD RAM
*
```

```
ISIOHC  MOVE.W  #B.PTMSK,SR      ;DISABLE INTS
        MOVE.W  (A7)+,P$UADR     ;POP TABLE INDEX
        MOVEM.L D0-A6,-(A7)      ;SAVE REGS
        LEA.L   ISIOB+ISCMDR,A0 ;POINT TO CMD RAM
        CMPI.W  #$100,P$UADR     ;BOARD #2 ??
        BLT.S   @000             ;N
        LEA.L   ISIOB2+ISCMDR,A0 ;Y
        SUB.W   #$100,P$UADR     ;ADJUST
*
@000    ADDA.W  P$UADR,A0        ;ADD BASE ADDR OFFSET
        CMPI.B  #$90,(A0)        ;DO WE HAVE A CHARACTER ?
        BEQ.S   @010             ;Y
        ADDA.W  #$20,A0          ;N, POINT TO OTHER CHANNEL
        CMPI.B  #$90,(A0)        ;CHARACTER ?
        BNE.S   @04              ;N, RETURN AND HOPE
*
@010    MOVE.W  2(A0),D2         ;GET CHARACTER
        MOVE.W  #$8000,(A0)
        MOVE.L  A0,D0
        ROR.W   #5,D0            ;GET PORT NUMBER ON BOARD
        AND.L   #$07,D0
        CMPA.L  #ISIOB2,A0
        BLT.S   @012
        OR.W    #$8,D0           ;SECOND BOARD BITS
@012    MOVE.W  P$ISINT,D1
        BTST    D0,D1            ;TEST HIGH WATER FLAG OF CHANNEL
        BNE.S   @014             ;HIGH WATER IF SET
        MOVE.W  #GETCHI,(A0)     ;SET NEW COMMAND
@014    MOVE.W  D1,P$ISINT
        MOVE.W  D2,D0            ;GET CHAR TO D0
        AND.W   #$0FF,D0         ;MASK IT
        MOVEA.L B$SRAM,A5        ;LOAD UP SYSRAM PTR
        MOVEA.L K1BEGN(A5),A1    ;LOAD KERNEL ENTRY POINT
        JMP     K2CHRI(A1)       ;TO KERNEL K2$CHRI
*
@04     MOVEM.L (A7)+,D0-A6      ;RESTORE REGS
        RTE                      ;RETURN & HOPE
        PAGE
*
*************************************************
*
        END UISIO1
```

### C.7 SYS68K/ISCSI-1 Driver Example

The following program is an example of a loadable driver for the ISCSI-1 board.

```
*       WSISCSI1:SR     26-APR-88
*****************************************************************
*                                                               *
*     FFFFF  OOO  RRRR   CCC  EEEEE    DDDD III  SSS  K   K     *
*     F     O   O R   R C   C E        D   D  I  S   S K  K      *
*     F     O   O R   R C     E        D   D  I  S     K K       *
*     FFFF  O   O RRRR  C     EEEE     D   D  I   SSS  KK        *
*     F     O   O R R   C     E        D   D  I      S K K       *
*     F     O   O R  R  C   C E        D   D  I  S   S K  K      *
*     F      OOO  R   R  CCC  EEEEE    DDDD III  SSS  K   K     *
*                                                               *
*                II   SSS    CCC    SSS   II    111             *
*                II  S   S  C   C  S   S  II   1111             *
*                II  S      C      S      II  11 11             *
*                II   SSS   C       SSS   II     11             *
*                II      S  C          S  II     11             *
*                II  S   S  C   C  S   S  II     11             *
*                II   SSS    CCC    SSS   II     11             *
*                                                               *
*****************************************************************
*
*       10-MAR-88  1.1  FIXED ERROR RETURN
*       26-APR-88  1.2  RETRY ON ERROR
*
WSISCSI1        IDNT    1.2     ISCSI1 DRIVER (installable)
*
        OPT     PDOS,ALT,ARS
        XREF    S$SRAM
        XDEF    SCSI1DRV
*
        INCLUDE FPARM:SR
*
*********************************************
*
*       COMMON DEFINITIONS
*
        IFUDF   FSTEP   :FSTEP  EQU  3   ;STEPRATE
*
W$EVNT  EQU     119             ;Suspension event
EVENTO  EQU     4*100           ;Disk timeout in TICS
        SECTION 14
```

```
*
*********************************************
*
* INSTALLABLE DRIVER TABLE
*
SCSI1DRV
START    DC.W    'W0'              ;IDENTIFIER
         BRA.S   WINIT             ;INITIALIZE DISK
         BRA.W   XDOF              ;DISK OFF
         NOP
         NOP
         NOP
         BRA.W   XREAD             ;READ SECTOR
         NOP
         NOP
         NOP
         BRA.W   XWRITE            ;WRITE SECTOR
         NOP
         NOP
         NOP
         DC.B    'FORCE ISCSI-1',0
         EVEN
         PAGE
 *
 ************************************************
 *       DISK INIT: Init the installed controller(s)
 *         and load up the parameter RAM as you find
 *         drives.
 *
 RL      REG     D0-A6
 *
 WINIT   MOVEM.L RL,-(A7)
         MOVE.L  $8+B.VEC,A4       ;SAVE BUS ERROR VECTOR
         MOVE.L  A7,A2             ;SAVE STACK
 *
         XLKT                      ;LOCK TASK BEFORE CHANGING BUSERR
                                   ;VECTOR
         PEA.L   @010(PC)
         MOVE.L  (A7)+,$8+B.VEC    ;SET NEW BUS ERROR
         LEA.L   P$SCSIF,A3        ;GET ADDRESS IN FFPARM
         LEA.L   ISCSIB,A1         ;GET BASE ADDRESS
         MOVE.W  #1,D0             ;GET COUNT
         MOVE.L  #$2000,D1         ;GET TEST ADDRESS
         MOVE.L  #0,D2             ;GET OFFSET
         CLR.B   (A3)              ;ASSUME NO BOARD
 * @002    TST.B   0(A1,D1.L)       ;BOARD PRESENT ? BUS ERROR IF NOT
         ADDQ.B  #1,(A3)           ;Y
         ADD.L   D2,A1             ;ADD OFFSET
         SUBQ.W  #1,D0             ;DECREMENT COUNT
         BNE.S @002
 *
 @010    MOVE.L  A2,A7             ;BUS ERROR TO HERE, RESTORE STACK
         MOVE.L  A4,$8+B.VEC       ;RESTORE BUS ERROR
         XULT                      ;UNLOCK TASK
```

```
*
        TST.B   P$SCSIF           ;CONTROLLER IN SYSTEM ??
        BEQ.S   @100              ;NO
        LEA.L   P$PARM,A4
        MOVEA.L A4,A6             ;SAVE P$FPARM
        CLR.L   (A4)+             ;NO FLOPPY FOR NOW
        CLR.L   (A4)+
        MOVEA.L A4,A5             ;SAVE P$WPARM
        CLR.L   (A4)+
        CLR.L   (A4)+
        CLR.L   (A4)+
        CLR.L   (A4)+             ;(A4) POINTS TO DRIVE 0 PARM AREA
        CLR.B   P$INTF            ;DON'T USE INTS OR DMA
*
* ISCSI DISK INIT
*
        LEA.L   ISCSI00(PC),A0  ;GET HANDLER ADDRESS
        MOVE.L  A0,P$DRW          ;DISK READ WRITE ADDRESS
        LEA.L   SCSIFORM(PC),A0 ;GET DISK FORMAT ADDRESS
        MOVE.L  A0,P$DFORM        ;SET DISK FORMAT ADDRESS
        BSR     INITSCSI          ;GO AND INIT THE CONTROLLER
        MOVEM.L (A7)+,RL
        MOVEQ.L #0,D0             ;GOOD RETURN
        RTS                       ;AND RETURN
@100    MOVEM.L (A7)+,RL
        MOVEQ.L #-1,D0            ;RETURN ERROR
        RTS                       ;AND RETURN
*
*********************************************
*       DRIVE NOT LEGAL
*
ERR100  MOVEQ.L #100,D0
        RTR
*
ERR101  MOVEQ.L #101,D0
        RTR
*
        EVEN
*************************************************
*       WRITE SECTOR
*
RL1     REG     D1-A6
*
XWRITE  MOVEQ.L #$1,D2            ;GET WRITE COMMANDS
        MOVE.L  D0,-(A7)          ;SAVE DISK NUMBER
        MOVEM.L RL1,-(A7)
        BSR.S   COMMON
        MOVEM.L (A7)+,RL1
        BNE.S   @001
        ADDA.L  #8,A7             ;ADJUST STACK (RA AND D0)
        MOVE.W  #4,-(A7)
        RTR
@001    CMPI.L  #100,D0           ;CHECK FOR ILLEGAL DRIVE
        BEQ.S   @002              ;Y
        ADDA.L  #8,A7             ;N, RETURN ERROR
        CLR.W   -(A7)
        RTR
@002    MOVE.L  (A7)+,D0          ;RESTORE DISK NUMBER
        CLR.W   -(A7)
        RTR
```

```
*
**************************************************
*       READ SECTOR
*
XREAD   MOVEQ.L #$0,D2            ;GET READ COMMANDS
        MOVE.L  D0,-(A7)          ;SAVE DISK NUMBER
        MOVEM.L RL1,-(A7)
        BSR.S   COMMON
        MOVEM.L (A7)+,RL1
        BNE.S   @001
        ADDA.L  #8,A7             ;ADJUST STACK (RA AND D0)
        MOVE.W  #4,-(A7)
        RTR
@001    CMPI.L  #100,D0           ;CHECK FOR ILLEGAL DRIVE
        BEQ.S   @002              ;Y
        ADDA.L  #8,A7             ;N, RETURN ERROR
        CLR.W   -(A7)
        RTR
@002    MOVE.L  (A7)+,D0          ;RESTORE DISK NUMBER
        CLR.W   -(A7)
        RTR
*
COMMON  CLR.W   -(A7)             ;PUSH .NE.
        ANDI.W  #$00FF,D0         ;DISK 0-255
        ANDI.L  #$FFFF,D1         ;MASK LOG SECTOR
        MOVEQ.L #1,D7             ;GET TRACK BIAS
        CMPI.W  #100,D0           ;BIAS?
          BLO.S @010              ;Y
        MOVEQ.L #0,D7             ;N, 0 BIAS
        SUBI.W  #100,D0           ;GET UNIT
*
@010    LEA.L   P$PARM,A1         ;POINT TO GENERAL PART ADDRESS TABLE
        MOVEQ.L #6,D3             ;6 TABLE ENTRIES
        CMPI.W  #100,D0           ;IS THIS A 200 DISK #?
          BLO.S @020              ;N, USE REGULAR PARTITION
*
*       DISK 200-239, SKIP TRACK MAPPING
*
        SUBI.W  #100,D0           ;Y, GET FAKE PART #
        MULU.W  #1,D0             ;MASK TO LOW WORD
        DIVU.W  #10,D0            ;GET D0 = PART | DRIVE #
        CMPI.W  #4,D0             ;0-3?
          BHS   ERR100            ;N, BAD DISK
        MOVE.L  D0,D7             ;SAVE BOTH
        CLR.W   D7                  ;D7 = GET 65526 * SECTION # (SECTOR
OFFSET)
        LSL.W   #2,D0             ;GET PART OFFSET *4
        TST.L   8(A1,D0.W)        ;DISK THERE?
          BEQ   ERR100            ;N
        MOVEA.L 8(A1,D0.W),A0     ;Y, POINT AT PARM TABLE
        ADD.L   D7,D1             ;D1 = GET TOTAL LOG SECTOR # WANTED
        BRA.S   @090              ;TO COMMON
*
```

```
*       REGULAR PARTITION : #0,N THROUGH #100,100+N
*
@020    CMPI.B  #2,D0           ;FLOPPY?
        BHS.S   @030            ;N, WINCH
        ADD.W   D7,D7           ;Y, BIAS FLOPPY BY 0 OR 2 TRACKS
*
*       (A1) = TABLE OF PART ADDRESSES
*       D3.W = TABLE COUNTER (# OF DRIVES ALLOCATED)
*
@030    TST.L   (A1)+           ;DRIVE EQUIPT?
        BEQ.S   @050            ;N
        MOVEA.L -4(A1),A0       ;Y, GET PART ADDRESS
        MOVEQ.L #6,D5           ;GET # OF BYTES PER PARTITION
        MULU.W  NPRT$(A0),D5   ;GET # OF PARTS ON DISK TIMES BYTES PER
        MOVE.W  D5,D4           ;SAVE BYTE OFFSET FOR PART TABLE
        LEA.L   PART$(A0),A3    ;POINT AT PARTS BASE/TOP/DISK#
*
@040    CMP.W   (A3)+,D0        ;SAME DISK #?
        BEQ.S   @060            ;Y, FOUND IT!!!
        ADDQ.W  #4,A3           ;SKIP OVER BASE TOP
        SUBQ.W  #6,D5           ;N, DONE ALL THESE PARTS?
        BGT.S   @040            ;N, LOOK AT ANOTHER
*
@050    SUBQ.W  #1,D3           ;Y, MORE DRIVES?
        BNE.S   @030            ;Y
        BRA     ERR100          ;N, ERROR 100
        PAGE
*
*       GET PHYSICAL SECTOR #:
*         D1.L = LOGICAL SECTOR
*         D2.B = COMMAND
*         (A3) = # OF CYLS IN PARTITION
*         D6.L = TOP TRACK
*         D7.W = TRACK BIAS
*         (A0) = DISK PARAMETERS
*
@060    ADD.W   (A3)+,D7        ;GET BASE TRACK #
        MOVEQ.L #0,D6
        MOVE.W  SPTK$(A0),D6    ;GET SPTK
        MULU.W  D6,D7           ;GET PHYS SECTOR OFFSET
        ADD.L   D7,D1           ;PHYSICAL SECTOR #
        MOVE.L  D1,D7           ;GET COPY
        DIVU.W  D6,D7           ;GET SECT|TRACK
        CMP.W   (A3),D7         ;TRACK TOO BIG?
          BHI   ERR101          ;Y, ERROR 101
        LEA.L   PART$(A0,D4.W),A1 ;N, GET POINTER TO BAD TRACK TABLE
        MOVEQ.L #0,D4           ;GET SECTOR ADJUSTOR
        TST.W   NBTK$(A0)       ;N, ANY BAD TRACKS?
          BEQ.S @090            ;N, SKIP ALL THIS
*
@070    TST.W   (A1)            ;ANY MORE TO CHECK?
        BEQ.S   @080            ;N, JUST ADD IN THESE
        CMP.W   (A1)+,D7        ;Y, IN RANGE?
        BLO.S   @080            ;N, JUST ADD IN THESE
        ADDQ.W  #1,D7           ;Y, UP LOGICAL TRACK #
        ADD.L   D6,D4           ;+ SPTK$(A0) = OFFSET IN SECTORS
        BRA.S   @070
```

```
*
@080    ADD.L   D4,D1           ;Y, ADD IN BAD TRACK OFFSET
*
*       ENTER FROM 200-209 PART
*
@090    MOVEQ.L #0,D3
        MOVE.B  CNTN$(A0),D3
        LSL.W   #2,D3           ;MUL BY 4
        MOVEQ.L #0,D0           ;CLEAR UPPER WORD
        MOVE.B  DSEL$(A0),D0    ;GET SELECT CODE
        BRA.L   ISCSI00
*
*
****************************************************************
*       SUSPEND ON TIMEOUT AND EVENT 119
*         OUT:  .EQ. if not timeout
*         USES: D0,D1
*
SUSPEND MOVE.L  #EVENTO,D0      ;GET 3 SECOND COUNT
        MOVEQ.L #$80,D1         ;GET LOCAL EVENT
        XDEV                    ;DELAY LOCAL EVENT X SECONDS
          BNE.S SUSPEND         ;DIDN'T GET IN, TRY AGAIN
        MOVE.W  #$8000+W$EVNT,D1 ;GET LOCAL/SUSPEND EVENT
        XSUI                   ;SUSPEND ON 119 AND LOCAL EVENT, WAIT...
        CMP.B   D0,D1           ;WAS IT EVENT 119?
        RTS
*
****************************************************************
*       Check Validity of and then move
*       Winch parms down to low memory
*
*       IN:     D6.W = CNTN$ | DSEL$
*               (A1) = DEFAULT TABLE
*               (A2) = DISK HEADER DATA
*               (A4) = NEXT AVAILABLE PARM AREA
*               (A5) = P$PARM ENTRY
*
*       OUT:    (A1) = PARAM RAM DATA
*               (A4) = NEXT AVAILABLE
*               updates A5
*
DOIT    CMPI.L  #'ME4U',(A2)+   ;IS WNERD INITED?
          BNE.S @010            ;N, USE DEFAULTS
        MOVE.W  (A2),D0         ;IS HEDS <= 0?
          BLE.S @010            ;Y, USE DEFAULTS
        SUBI.W  #16,D0          ;IS HEDS > 16?
          BGT.S @010            ;Y, USE DEFAULTS
        TST.W   SPTK$(A2)       ;PDOS SECTORS PER TRAKS NON-ZERO?
          BEQ.S @010            ;N, USE DEFAULTS
        MOVEA.L A2,A1           ;USE DISK HEADER DATA
*
@010    MOVE.W  NPRT$(A1),D0    ;GET # OF PARTS
        MULU.W  #3,D0           ;= # OF WORDS
        ADD.W   NBTK$(A1),D0    ;+ BAD TRACK ENTRIES
        ADDI.W  #PART$/2+1-1,D0 ;+ 8+1 TERMIN HEADER INFO WORDS
        MOVE.L  A4,(A5)         ;SAVE ADDRESS OF THIS PARM TABLE
```

```
*
@020    MOVE.W  (A1)+,(A4)+
          DBF   D0,@020
        MOVEA.L (A5)+,A1          ;GET PARM ADDRESS
        MOVE.W  D6,CNTN$(A1)      ;SET CONTROLLER & DSEL
        RTS
        PAGE
****************************************
*       DISK OFF ROUTINE:
*
XDOF    RTS
        PAGE
*
 *******************************************************************
*
* DRIVER FOR FORCE ISCSI-1 DISK CONTROLLER
* THIS CONTROLLER IS CALLED FROM THE STANDARD FORCE DISK CONTROLLER
* HEADER.
*
* UPDATE SCHEDULE:
* 04-FEB-87  1.0  M.S.  INITIAL VERSION
* 18-FEB-87  1.1  M.S.  Reduced default number of tracks on Micropolis
*                       winchester and the number of floppy partitions
*                       supported by default.
* 10-MAR-87  1.2  M.S.  Wait at INIT for SYSFAIL to disappear in
*                       the ISCSI-1 status register
*
*
        IFUDF   RETRY  :RETRY   EQU  1 ; RETRY ON ERROR
 *******************************************************************
*         --- ISCSI-1 CODE ---
*
*         D0.B = SELECT CODE BYTE      (TARGET ID | LUN)
*         D1.L = LOGICAL SECTOR #      (BLOCK NUMBER)
*         D2.L = READ/WRITE COMMAND    (0/1)
*         (A2) = DATA ADDRESS
*         A5   = B$SRAM ADDRESS
*
ISCSINIT CLR.W  -(A7)             ;SET .NE. FOR ENTRY FROM W$XDIT
*
* BUILD COMMAND IN D2.L: $2000=READ, $2200=WRITE
*
ISCSI00 ADD.W   #$20,D2           ;MAKE 20,21
        CMPI.B  #$20,D2           ;READ ?
          BEQ.S @000              ;Y
        ADDQ.L  #1,D2             ;N, MAKE WRITE
@000    LSL.L   #8,D2             ;MAKE $2000/$2200
*
```

```
* BUILD COMMAND RAM IN D-REGISTERS
*    D3.L = CMD | NUMBER OF BLOCKS(=1)
*    D4.L = BLOCK ADDRESS
*    D5.L = DATA ADDRESS IN ISCSI DPR
*    D6.L = LUN | 00
*
        LEA.L   ISCCMD1,A1       ;GET ISCSI-1 BASE ADDRESS
        CLR.W   $E(A1)           ;RESET RETRY FLAG (MULTIPROC)
        MOVEQ.L #1,D3            ;1 BLOCK
        SWAP    D3               ;COUNT | 00
        MOVE.W  D2,D3            ;GET COMMAND: COUNT | COMMAND
        LSR.W   #8,D3            ;GET ISCSI COMMAND
        BTST.B  #ISCSI,P$INTF    ;USING ISCSI INTS?
          BEQ.S @010             ;N, SKIP EVENT RESET
        OR.W    #$1000,D3        ;SET INTERRUPT BIT
        BCLR.B  #~W$EVNT,W$EVNT/8+EVTB.+S$SRAM ;RESET EVENT 119
@010    CLR.L   D6
        MOVE.B  D0,D6            ;GET ID AND LUN
        ROR.L   #4,D6
        ROR.W   #7,D6            ;GET
LLLL.0000.0000.0000|0000.III0.0000.0000
        OR.W    D6,D3            ;PUT ID INTO COMMAND
        SWAP    D6
        ROL.W   #4,D6            ;D6.W = 0000LLLL
        MOVE.L  D1,D4
        LEA.L   ISCIOB1,A4
        MOVE.L  A4,D5            ;GET I/O BUFFER ADDRESS
*
* COMMAND BLOCK IS COMPLETE NOW, GO TO READ OR WRITE COMMAND
*
        MOVE.W  #64-1,D7         ;GET LONG_WORD COUNT
@020    TST.B   (A1)             ;WAIT HERE IF CONTROLLER BUSY
          BPL.S @020             ;CONTROLLER BUSY (??)
        CMPI.B  #$20,D3          ;READ OR WRITE ?
          BEQ.S @050             ;IT IS READ
*
* IT IS WRITE -- CHECK DMA AND TRANSFER DATA
*
@040    BTST.B  #DSCSI,P$INTF    ;USE DMA FOR TRANSFER ?
          BEQ.S @048             ;N
*
* WE USE DMA
*
        MOVE.L  P$DMAC,A3        ;POINT TO CHIP
        MOVE.B  #$FF,DCSR(A3)    ;RESET DMA
        MOVE.L  A2,DMAR(A3)      ;SET MEMORY ADDRESS
        MOVE.L  D5,DDAR(A3)      ;SET DEVICE ADDRESS
        MOVE.W  #128,DMTC(A3)    ;DO 256 BYTES TRANSFER
        MOVE.B  #$11,DOCR(A3)    ;DO MEM TO DEV ON MAX SPEED
        MOVE.B  #$80,DCCR(A3)    ;GO !!
@044    BTST.B  #7,DCSR(A3)      ;POLL FOR COMPLETION
        BEQ.S   @044             ;WAIT
        BTST.B  #4,DCSR(A3)      ;ERROR ?
        BEQ.S   @050             ;N
        MOVEQ.L #107,D0          ;ERROR 107 = DMA ERROR
        BRA     ISCOUT
```

```
*
* CPU DOES THE TRANSFER
*
@048    MOVE.L  (A2)+,(A4)+      ;OUTPUT DATA TO CONTROLLER
          DBF   D7,@048          ;LOOP
*
* DMA AND POLLING,  WRITE, COME HERE AFTER DATA TRANSFER
*
@050    SWAP    D3
        MOVE.W  D6,$C(A1)        ;SET LUN
@054    MOVEM.L D3-D5,(A1)       ;LOAD UP COMMAND RAM
        BTST.B  #ISCSI,P$INTF    ;USE ISCSI INTS?
          BEQ.S @080             ;N, JUST POLL BUSY
        BSR     SUSPEND          ;Y, SUSPEND ON EV 119, TIMEOUT?
          BEQ.S @080             ;N, GET STATUS AND DROP INTO FINISH
        BCLR.B  #ISCSI,P$INTF    ;Y, DISABLE INTS FOR NEXT TIME
*
@080    CLR.L   D1
@082    MOVE.W  (A1),D1          ;GET STATUS
          BPL.S @082             ;LOOP, STILL BUSY
        SWAP    D3               ;GET COMMAND
        TST.B   D1               ;ERROR ??
*
* THE FOLLOWING LINE IS USED IF RETRY IS DISABLED
*
        IFEQ    RETRY
          BNE.S ISCSIE           ;Y
        ENDC
*
* THE FOLLOWING CODE IS USED FOR RETRY
*
        IFNE    RETRY
          BEQ.S @088             ;N
        MOVE.W  D3,D0
        AND.W   #$0E00,D0        ;MASK
        CMPI.W  #$E00,D0         ;ID 7 ?
          BEQ.W ISCSIE           ;Y, DO ERROR HANDLING
        CMPI.B  #$F,D1           ;EXTENDED ERROR
          BNE.W ISCSIE
        CMPI.W  #$109,2(A1)      ;SCSI CHECK CONDITION ?
          BNE.W ISCSIE           ;N
        TAS.B   $E(A1)           ;Y, RETRY ?
          BNE.W ISCSIE           ;N
        MOVEM.L D0,-(A7)         ;Y,SAVE REGISTER
@087    MOVE.L  #$2600000,D0
@087A   SUBQ.L  #1,D0
          BNE.S @087A
        MOVEM.L (A7)+,D0         ;RELOAD REGISTER
        SWAP    D3               ;
        BRA.S   @054             ;DO COMMAND AGAIN
        ENDC
*
@088    CMPI.B  #$20,D3          ;WAS IT A READ?
          BNE.S @100             ;N, TO COMMON
```

```
*
* IT IS READ -- CHECK DMA AND TRANSFER DATA
*
@090    BTST.B  #DSCSI,P$INTF   ;USE DMA FOR TRANSFER ?
          BEQ.S @098            ;N
*
* WE USE DMA
*
        MOVE.L  P$DMAC,A3       ;POINT TO CHIP
        MOVE.B  #$FF,DCSR(A3)   ;RESET DMA
        MOVE.L  A2,DMAR(A3)     ;SET MEMORY ADDRESS
        MOVE.L  4(A1),DDAR(A3)  ;SET DEVICE ADDRESS FROM RETURN CODE
        MOVE.W  #128,DMTC(A3)   ;DO 256 BYTES TRANSFER
        MOVE.B  #$91,DOCR(A3)   ;DO DEV TO MEM ON MAX SPEED
        MOVE.B  #$80,DCCR(A3)   ;GO !!
@094    BTST.B  #7,DCSR(A3)     ;POLL FOR COMPLETION
        BEQ.S   @094            ;WAIT
        BTST.B  #4,DCSR(A3)     ;ERROR ?
        BEQ.S   @100            ;N
        MOVEQ.L #107,D0         ;ERROR 107 = DMA ERROR
        BRA.S   ISCOUT
*
* CPU DOES THE TRANSFER
*
@098    MOVEA.L 4(A1),A4        ;GET DATA ADDRESS
@099    MOVE.L  (A4)+,(A2)+     ;READ FROM BUFFER
          DBF   D7,@099         ;LOOP
*
* COMMON TERMINATION CODE
*
@100    ADDQ.W  #4,(A7)         ;SET .EQ.
*
* RETURN WITH ERROR IN D0
*
ISCOUT
            RTR
*
* ISCSI-1 ERROR HANDLER
*
ISCSIE  CMPI.B  #$F,D1          ;EXTENDED ERROR ??
         BNE.S  @010            ;N
        MOVE.W  2(A1),D1        ;Y, GET EXTENDED ERROR NUMBER
        CLR.L   D0
        MOVE.B  D1,D0
        LSR.W   #8,D1
        CMPI.B  #3,D1           ;FLOPPY ERROR ?
         BNE.S  @002            ;N, ERROR 1-7
        ADD.W   #10,D0          ;Y, ERROR 11-17

@002    ADD.W   #160,D0         ;SCSI = 110-120
        BRA.S   ISCOUT
*
@010    ADD.W   #146,D1         ;STANDARD ERROR
        MOVE.L  D1,D0
        BRA.S   ISCOUT
        PAGE
```

```
 ****************************************************************
*       SYS68K/SCSI-1 DISK INIT
*
*  IN: (A4) = DISK 0 PARAMETER TABLE
*      (A5) = P$WPARM AREA
*      (A6) = P$FPARM AREA
*
INITSCSI
*
* INIT DMAC
*
        TST.W   P$DMAC          ;DO WE HAVE A DMA ??
         BEQ.S  @000            ;N
        MOVE.L  P$DMAC,A1       ;Y, GET BASE
        MOVE.B  #$FF,DCSR(A1)   ;RESET CSR
        MOVE.B  #$08,DDCR(A1)   ;SET BURST, 8-BIT 68000 DEV
                                    (ALSO IF 16-BIT!!!)
        MOVE.B  #5,DSCR(A1)     ;MEM AND DEVICE COUNTS UP
        MOVE.B  #5,DMFC(A1)     ;SUPERVISOR DATA ACCESS
        MOVE.B  #5,DDFC(A1)     ;SUPERVISOR DATA ACCESS
        MOVE.B  #0,DBFC(A1)     ;BFC IS DON'T CARE
*
* INIT BIM
*
@000    LEA.L   ISCBIM,A0       ;POINT TO BIM
        MOVE.B  #W$EVNT,ISCBV0(A0) ;SET COMPLETE VECTOR
        MOVE.B  #W$EVNT,ISCBV1(A0) ;ERROR VECTOR IS SAME
        MOVE.B  #$54,ISCBC0(A0) ;LEVEL 4
        MOVE.B  #$54,ISCBC1(A0) ;ERROR IS SAME
*
* SET OFFSET TO BASE ADDRESS
*
        MOVE.L  A0,D0           ;GET ADDRESS OF BOARD
        AND.L   #$FFFE0000,D0   ;MASK IT
        LEA.L   ISCCMD1,A0
@010    MOVE.W  ISCSIB,D1       ;READ STATUS REGISTER
        ROR.W   #8,D1
        BTST    #2,D1           ;TEST SYSFAIL FLAG, WAIT HERE IF SET
        BEQ.S   @010
@011    TST.B   (A0)            ;WAIT HERE IF BUSY
         BPL.S  @011
        MOVE.L  D0,4(A0)        ;SET BASE ADDRESS
        MOVE.W  #$3,(A0)        ;WRITE COMMAND
@012    TST.B   (A0)            ;WAIT UNTIL DONE
         BPL.S  @012
*
```

```
* INIT 3 SCSI WINIS (ID 0,1,2)
*
        LEA.L   INIDAT(PC),A1   ;POINT TO DATA
        LEA.L   ISCIOB1,A2      ;POINT TO I/O BUFFER
        MOVE.L  (A1)+,(A2)+     ;3 LONGS
        MOVE.L  (A1)+,(A2)+
        MOVE.L  (A1)+,(A2)+
        LEA.L   INIINS(PC),A1   ;GET INIT INSTRUCTIONS
@020    MOVEM.L (A1)+,D3-D6     ;GET THEM
        MOVE.W  D6,$C(A0)       ;SET LUN
        MOVEM.L D3-D5,(A0)      ;WRITE TO ISCSI CMD2
@022    TST.B   (A0)            ;WAIT FOR COMMAND TO COMPLETE
         BPL.S  @022
        TST.W   (A1)            ;END OF TABLE ?
         BPL.S  @020            ;N
*
* UNIT PARMS ARE SET NOW FOR 3 WINIS, FLOPPIES USE THE DEFAULT
* NOW WE SET THE FLOPPY PARTITIONS AND TRY TO READ THE HEADER
* INFORMATION OF THE WINIS.
*
        MOVE.W  #$0373,D6       ;CNTR | DSEL FOR FLOPPY
        LEA.L   FLPARM(PC),A1   ;GET FLOPPY TABLE
        MOVEA.L A1,A2           ;GET A DUMMY POINTER
        EXG     A5,A6           ;A5 = P$FPARM
        BSR     DOIT            ;LOADUP FLOPPY PARMS
        ADDQ.L  #1,D6           ;SECOND FLOPPY DRIVE
        LEA.L   FL1PARM(PC),A1  ;GET FLOPPY TABLE FOR SECOND DRIVE
        BSR     DOIT            ;LOADUP PARMS OF SECOND FLOPPY
        EXG     A5,A6           ;GET WINI PARM AREA
*
        MOVE.W  #$300,D6        ;CNT | ID
@040    LEA.L   P$BLOCK,A2      ;DISK DATA SHOULD GO THERE
        MOVE.B  D6,D0           ;WE START WITH ID 0
        MOVEQ.L #0,D1           ;READ BLOCK 0
        MOVEQ.L #$0,D2          ;READ COMMAND
        MOVEM.L D6/A2/A4/A5,-(A7)       ;SAVE THESE PARMS
        CLR.L   (A2)            ;DESTROY MESSAGE
        LEA.L   B$SRAM,A5
        BSR     ISCSINIT        ;READ BLOCK 0
        MOVEM.L (A7)+,D6/A2/A4/A5
          BNE.S @054            ;SKIP IF READ ERROR
        LEA.L   WIPARM(PC),A1   ;GET DEFAULT WINI PARMS
        BSR     DOIT            ;LOAD UP PARMS AS DRIVE FOUND
@054    ADD.W   #$10,D6         ;INCREMENT LUN
        CMPI.B  #$30,D6         ;DONE 3 WINIS ?
          BNE.S @040            ;N
*
@050
        TST.W   P$DMAC          ;DO WE HAVE A DMA ?
        BEQ.S   @052            ;N
        BSET    #DSCSI,P$INTF   ;Y, USE IT
@052    BSET    #ISCSI,P$INTF   ;ENABLE INTS
        RTS
        PAGE
```

```
 *****************************************************************
*       ISCSI-1 DISK FORMATTER
* THIS FUNCTION DOES A WINCHESTER FORMAT WITH THE INSTALLED UNIT
* PARAMETERS.
*
* IN:   0(A7).L = RETURN ADDRESS
*       4(A7).W = ID | LUN
*   OR AT ENTRY ADDRESS + 4:
*       D0.W    = ID| LUN
*
* OUT:  D0 = ERROR CODE OR 0 IF NO ERROR
*       SR = .NE.       OR   .EQ.
*
* NO REGISTERS ARE DESTROYED (EXCEPT D0 FOR STATUS RETURN)
*
SCSIFORM MOVE.W 4(A7),D0        ;GET PARMS
SCSIF1  MOVEM.L D1/D2/A0,-(A7)  ;SAVE REGISTERS
        LEA.L   ISCCMD1,A0      ;GET CONTROLLER ADDRESS
        CLR.W   $E(A0)          ;CLEAR FLAG
        MOVE.W  D0,D2           ;GET A COPY
@000    MOVE.W  D2,D0           ;GET ID | LUN
        MOVE.W  D0,D1
        AND.W   #$0FF,D0        ;D0.W = LUN
        AND.W   #$0F00,D1       ;D1.W = ID | 00
        ROL.W   #1,D1           ;POSITION ID
        OR.W    #$57,D1         ;OR-IN FORMAT COMMAND
*
@002    TST.B   (A0)            ;BUSY ?
          BPL.S @002            ;Y, WAIT
        MOVE.W  D0,$C(A0)       ;WRITE LUN
        MOVE.W  D1,(A0)         ;WRITE COMMAND AND ID
@006    MOVE.W  (A0),D0         ;GET STATUS, BUSY ?
          BPL.S @006            ;Y ,WAIT
        AND.W   #$0FF,D0        ;MASK
          BEQ.S @020            ;NO ERROR
*       TAS.B   $E(A0)          ;ERROR BEFORE ??
          BEQ.S @000            ;N, RETRY
        CMPI.W  #$0F,D0         ;EXTENDED ERROR ?
          BNE.S @020            ;N
        MOVE.W  2(A0),D0        ;Y, GET ERROR
@020    MOVEM.L (A7)+,D1/D2/A0
        RTS
        PAGE
*******************************************************************
*       DATA FOR UNIT PARAMETERS
*
INIDAT  DC.L    0
        DC.L    0                ; USE DEFAULT FOR THOSE
        DC.B    0                ; HARD DISK ASSUMED
        DC.B    0                ; 256 BYTES/SECTOR
        DC.B    1                ; HASHING ENABLED
        DC.B    0                ; DUMMY FOR EVEN ADDRESS
*
```

```
*       COMMANDS TO SET UNIT PARAMETERS FOR 3 WINIS
*
INIINS  DC.B    0               ; ID 0
        DC.B    $0A             ; OPCODE
        DC.W    0               ; NOT USED
        DC.L    ISCIOB1         ; ADDRESS OF FIRST I/O BUFFER
        DC.L    0               ; NOT USED
        DC.W    0               ; LUN 0
        DC.W    0               ; NOT USED
*
        DC.B    2               ; ID 1
        DC.B    $0A             ; OPCODE
        DC.W    0               ; NOT USED
        DC.L    ISCIOB1         ; ADDRESS OF FIRST I/O BUFFER
        DC.L    0               ; NOT USED
        DC.W    0               ; LUN 0
        DC.W    0               ; NOT USED
*
        DC.B    4               ; ID 2
        DC.B    $0A             ; OPCODE
        DC.W    0               ; NOT USED
        DC.L    ISCIOB1         ; ADDRESS OF FIRST I/O BUFFER
        DC.L    0               ; NOT USED
        DC.W    0               ; LUN 0
        DC.W    0               ; NOT USED
*
        DC.W    $FFFF           ; END OF TABLE
*
        PAGE
*****************************************************************
*
*       DEFAULT ISCSI-1 HEADER PARTITIONS
*
* MOST OF THE VALUES IN THE TABLE ARE DUMMIES AS THE ACTUAL
* NUMBER OF CYLS, HEADS, SECTORS/TRACK ARE NOT USED ON SCSI
* WINCHESTER DRIVES.
* THE FOLLOWING IS VALID FOR THE MICROPOLIS WINCHESTER 1375. THIS
* DEVICE HAS MORE THAN 513000 BLOCKS WHEN FORMATTED WITH 256 BYTES
* PER SECTOR. THE TRANSLATION FOR PDOS USES 512000 BLOCKS.
*
*
WIPARM  DC.W    16              ; HEDS
        DC.W    1000            ; CYLS
        DC.W    32              ; BPT
        DC.W    256             ; BPB
        DC.W    0               ; SHIP
        DC.W    32              ; SPT
        DC.W    22              ; # OF PARTS
        DC.W    0               ; BAD TRACKS, NOT USED !!
        DC.W    $0300           ;CNTN & DSEL
        DC.W    0               ;STEP
        DC.W    0               ;REDO WRT CURR
        DC.W    0               ;WRT PERCOMP
        DC.W    2,0,1499        ;WINI PART 1 ...
        DC.W    3,1500,2999
        DC.W    4,3000,4499
        DC.W    5,4500,5999
        DC.W    6,6000,7499
        DC.W    7,7500,8999
        DC.W    9,9000,10499
```

```
        DC.W    10,10500,11999
        DC.W    11,12000,13499
        DC.W    12,13500,14999  ;..WINI PART 10
        DC.W    13,15000,15079  ;FLOPPY PART 1 ...
        DC.W    14,15080,15159
        DC.W    15,15160,15239
        DC.W    16,15240,15319
        DC.W    17,15320,15399
        DC.W    18,15400,15479
        DC.W    19,15480,15559
        DC.W    20,15560,15639
        DC.W    21,15640,15719
        DC.W    22,15720,15799
        DC.W    23,15800,15879
        DC.W    24,15880,15959  ;..FLOPPY PART 12
        DC.W    0
*
*************************************
*       FAKE FLOPPY PARTITION TABLE
*
FLPARM  DC.W    2               ;FLOPPY HEDS
        DC.W    80              ;CYLS
        DC.W    16              ;PHYS BPT BLOCKS PER TRACK
        DC.W    256             ;PHYS BPB BYTES PER BLOCK
        DC.W    0               ;SHIP
        DC.W    16              ;# PDOS SPT SECTORS PER TRACK
        DC.W    1               ;# PARTS
        DC.W    0               ;# BAD TRKS
        DC.W    $0173           ;CNTN & DSEL
        DC.W    0               ;STEP
        DC.W    0               ;REDU WRT CURR
        DC.W    0               ;WRT PERCOMP
        DC.W    0               ;DISK #0
        DC.W    0,159           ;PARTITION 0
        DC.W    0
*
FL1PARM DC.W    2               ;FLOPPY HEDS
        DC.W    80              ;CYLS
        DC.W    16              ;PHYS BPT BLOCKS PER TRACK
        DC.W    256             ;PHYS BPB BYTES PER BLOCK
        DC.W    0               ;SHIP
        DC.W    16              ;# PDOS SPT SECTORS PER TRACK
        DC.W    1               ;# PARTS
        DC.W    0               ;# BAD TRKS
        DC.W    $0174           ;CNTN & DSEL
        DC.W    0               ;STEP
        DC.W    0               ;REDU WRT CURR
        DC.W    0               ;WRT PERCOMP
        DC.W    1               ;DISK #1
        DC.W    0,159           ;PARTITION 1
        DC.W    0
*
******************** END OF ISCSI-1 DRIVER *************************

        EVEN
        END
```

## C.8 SYS68K/WFC-1 Driver Example

The following program is an example of a loadable driver for the WFC-1 board.

```
*       WSWFC1:SR       10-MAR-88
**************************************************************
*                                                            *
*    FFFFF  OOO   RRRR   CCC  EEEEE   DDDD  III  SSS  K   K  *
*    F     O   O R   R C   C E         D   D  I  S   S K  K   *
*    F     O   O R   R C     E         D   D  I  S     K K    *
*    FFFF  O   O RRRR  C     EEEE      D   D  I   SSS  KK     *
*    F     O   O R R   C     E         D   D  I      S K K    *
*    F     O   O R  R  C   C E         D   D  I  S   S K  K   *
*    F      OOO  R   R  CCC  EEEEE   DDDD  III  SSS  K   K  *
*                                                            *
*              W       W  FFFFF   CCC     111               *
*              W       W  F      C   C   1111               *
*              W       W  F      C      11 11               *
*              W   W   W  FFFF   C         11               *
*              W  W W  W  F      C         11               *
*              W W   W W  F      C   C     11               *
*              WW     WW  F       CCC      11               *
*                                                            *
**************************************************************
*
*       10-MAR-88  1.1  FIXED ERROR RETURN
*
WSWFC1  IDNT       1.1  WFC1 DRIVER (installable)
*
        OPT     PDOS,ALT,ARS
        XDEF    WFCDRV
*
        INCLUDE  FPARM:SR              ; 16-BIT VERSION
        PAGE
********************************************
*
*       COMMON DEFINITIONS
*
        IFUDF   FSTEP   :FSTEP EQU 3   ;STEPRATE
*
W$EVNT  EQU     119                    ;Suspension event
EVENTO  EQU     4*100                  ;Disk timeout in TICS
        SECTION 14
```

```
********************************************
*
* INSTALLABLE DRIVER TABLE
*
WFCDRV
START   DC.W    'W0'               ;IDENTIFIER
        BRA.S   WINIT              ;INITIALIZE DISK
        BRA.W   XDOF               ;DISK OFF
        NOP
        NOP
        NOP
        BRA.W   XREAD              ;READ SECTOR
        NOP
        NOP
        NOP
        BRA.W   XWRITE             ;WRITE SECTOR
        NOP
        NOP
        NOP
        DC.B    'FORCE WFC-1',0
        EVEN
 *
 *********************************************
 *       DISK INIT: Init the installed controller(s)
 *       and load up the parameter RAM as you find
 *       drives.
 *
 RL      REG     D0-A6
 *
 WINIT   MOVEM.L RL,-(A7)
         MOVE.L  $8+B.VEC,A4       ;SAVE BUS ERROR VECTOR
         MOVE.L  A7,A2             ;SAVE STACK
 *
         XLKT                     ;LOCK TASK BEFORE CHANGING BUSERR VECTOR
         PEA.L   @010(PC)
         MOVE.L  (A7)+,$8+B.VEC    ;SET NEW BUS ERROR
         LEA.L   P$WFCF,A3         ;GET ADDRESS IN FPARM
         LEA.L   WFCBASE,A1        ;GET BASE ADDRESS
         MOVE.W  #1,D0             ;GET COUNT
         MOVE.L  #SDHD,D1          ;GET TEST ADDRESS
         MOVE.L  #$10,D2           ;GET OFFSET
         CLR.B   (A3)              ;ASSUME NO BOARD
* @001   TST.B   0(A1,D1.L)        ;BOARD PRESENT ? BUS ERROR IF NOT
         ADDQ.B  #1,(A3)           ;Y
         ADD.L   D2,A1             ;ADD OFFSET
         SUBQ.W  #1,D0             ;DECREMENT COUNT
         BNE.S @001
*
 @010    MOVE.L  A2,A7             ;BUS ERROR TO HERE, RESTORE STACK
         MOVE.L  A4,$8+B.VEC       ;RESTORE BUS ERROR
         XULT                      ;UNLOCK TASK
```

```
*
        TST.B   P$WFCF          ;CONTROLLER IN SYSTEM ??
        BEQ.S   @100            ;NO
        LEA.L   P$PARM,A4
        MOVEA.L A4,A6           ;SAVE P$FPARM
        CLR.L   (A4)+           ;NO FLOPPY FOR NOW
        CLR.L   (A4)+
        MOVEA.L A4,A5           ;SAVE P$WPARM
        CLR.L   (A4)+
        CLR.L   (A4)+
        CLR.L   (A4)+
        CLR.L   (A4)+           ;(A4) POINTS TO DRIVE 0 PARM AREA
        CLR.B   P$INTF          ;DON'T USE INTS OR DMA
*
* WFC-1 DISK INIT
*
        LEA.L   WFC00(PC),A0    ;GET HANDLER ADDRESS
        MOVE.L  A0,P$DRW        ;DISK READ WRITE ADDRESS
        LEA.L   WFCFORM(PC),A0  ;GET DISK FORMAT ADDRESS
        MOVE.L  A0,P$DFORM      ;SET DISK FORMAT ADDRESS
        BSR     INITWFC         ;GO AND INIT THE CONTROLLER
*
        MOVEM.L (A7)+,RL
        MOVEQ.L #0,D0           ;GOOD RETURN
        RTS                     ;AND RETURN
@100    MOVEM.L (A7)+,RL
        MOVEQ.L #-1,D0          ;RETURN ERROR
        RTS                     ;AND RETURN
*
*
*********************************************
*       DRIVE NOT LEGAL
*
ERR100  MOVEQ.L #100,D0
        RTR
*
ERR101  MOVEQ.L #101,D0
        RTR
*
        EVEN
*************************************************
*       WRITE SECTOR
*
RL1     REG     D1-A6
*
XWRITE  MOVEQ.L #$1,D2          ;GET WRITE COMMANDS
        MOVE.L  D0,-(A7)
        MOVEM.L RL1,-(A7)
        BSR.S   COMMON
        MOVEM.L (A7)+,RL1
        BNE.S   @001
        ADDA.L  #8,A7           ;ADJUST STACK (RA AND D0)
        MOVE.W  #4,-(A7)
        RTR
@001    CMPI.L  #100,D0         ;CHECK FOR ILLEGAL DRIVE
        BEQ.S   @002            ;Y
        ADDA.L  #8,A7           ;N, RETURN ERROR
        CLR.W   -(A7)
        RTR
```

```
@002    MOVE.L  (A7)+,D0        ;RESTORE DISK NUMBER
        CLR.W   -(A7)
        RTR
*
*************************************************
*       READ SECTOR
*
XREAD   MOVEQ.L #$0,D2          ;GET READ COMMANDS
        MOVE.L  D0,-(A7)
        MOVEM.L RL1,-(A7)
        BSR.S   COMMON
        MOVEM.L (A7)+,RL1
        BNE.S   @001
        ADDA.L  #8,A7           ;ADJUST STACK (RA AND D0)
        MOVE.W  #4,-(A7)
        RTR
@001    CMPI.L  #100,D0         ;CHECK FOR ILLEGAL DRIVE
        BEQ.S   @002            ;Y
        ADDA.L  #8,A7           ;N, RETURN ERROR
        CLR.W   -(A7)
        RTR
@002    MOVE.L  (A7)+,D0        ;RESTORE DISK NUMBER
        CLR.W   -(A7)
        RTR
*
COMMON  CLR.W   -(A7)           ;PUSH .NE.
        ANDI.W  #$00FF,D0       ;DISK 0-255
        ANDI.L  #$FFFF,D1       ;MASK LOG SECTOR
        MOVEQ.L #1,D7           ;GET TRACK BIAS
        CMPI.W  #100,D0         ;BIAS?
          BLO.S @010            ;Y
        MOVEQ.L #0,D7           ;N, 0 BIAS
        SUBI.W  #100,D0         ;GET UNIT
*
@010    LEA.L   P$PARM,A1       ;POINT TO GENERAL PART ADDRESS TABLE
        MOVEQ.L #6,D3           ;6 TABLE ENTRIES
        CMPI.W  #100,D0         ;IS THIS A 200 DISK #?
          BLO.S @020            ;N, USE REGULAR PARTITION
        PAGE
*
*       DISK 200-239, SKIP TRACK MAPPING
*
        SUBI.W  #100,D0         ;Y, GET FAKE PART #
        MULU.W  #1,D0           ;MASK TO LOW WORD
        DIVU.W  #10,D0          ;GET D0 = PART | DRIVE #
        CMPI.W  #4,D0           ;0-3?
          BHS   ERR100          ;N, BAD DISK
        MOVE.L  D0,D7           ;SAVE BOTH
        CLR.W   D7             ;D7=GET 65526*SECTION# (SECTOR OFFSET)
        LSL.W   #2,D0           ;GET PART OFFSET *4
        TST.L   8(A1,D0.W)      ;DISK THERE?
          BEQ   ERR100          ;N
        MOVEA.L 8(A1,D0.W),A0   ;Y, POINT AT PARM TABLE
        ADD.L   D7,D1           ;D1 = GET TOTAL LOG SECTOR # WANTED
        BRA.S   @090            ;TO COMMON
*
*       REGULAR PARTITION : #0,N THROUGH #100,100+N
*
```

```
@020    CMPI.B  #2,D0           ;FLOPPY?
          BHS.S @030            ;N, WINCH
        ADD.W   D7,D7           ;Y, BIAS FLOPPY BY 0 OR 2 TRACKS
*
*       (A1) = TABLE OF PART ADDRESSES
*       D3.W = TABLE COUNTER (# OF DRIVES ALLOCATED)
*
@030    TST.L   (A1)+           ;DRIVE EQUIPT?
          BEQ.S @050            ;N
        MOVEA.L -4(A1),A0       ;Y, GET PART ADDRESS
        MOVEQ.L #6,D5           ;GET # OF BYTES PER PARTITION
        MULU.W  NPRT$(A0),D5   ;GET # OF PARTS ON DISK TIMES BYTES PER
        MOVE.W  D5,D4           ;SAVE BYTE OFFSET FOR PART TABLE
        LEA.L   PART$(A0),A3    ;POINT AT PARTS BASE/TOP/DISK#
*
@040    CMP.W   (A3)+,D0        ;SAME DISK #?
          BEQ.S @060            ;Y, FOUND IT!!!
        ADDQ.W  #4,A3           ;SKIP OVER BASE TOP
        SUBQ.W  #6,D5           ;N, DONE ALL THESE PARTS?
          BGT.S @040            ;N, LOOK AT ANOTHER
*
@050    SUBQ.W  #1,D3           ;Y, MORE DRIVES?
          BNE.S @030            ;Y
        BRA     ERR100          ;N, ERROR 100
        PAGE
*
*       GET PHYSICAL SECTOR #:
*         D1.L = LOGICAL SECTOR
*         D2.B = COMMAND
*         (A3) = # OF CYLS IN PARTITION
*         D6.L = TOP TRACK
*         D7.W = TRACK BIAS
*         (A0) = DISK PARAMETERS
*
@060    ADD.W   (A3)+,D7        ;GET BASE TRACK #
        MOVEQ.L #0,D6
        MOVE.W  SPTK$(A0),D6    ;GET SPTK
        MULU.W  D6,D7           ;GET PHYS SECTOR OFFSET
        ADD.L   D7,D1           ;PHYSICAL SECTOR #
        MOVE.L  D1,D7           ;GET COPY
        DIVU.W  D6,D7           ;GET SECT|TRACK
        CMP.W   (A3),D7         ;TRACK TOO BIG?
          BHI   ERR101          ;Y, ERROR 101
        LEA.L   PART$(A0,D4.W),A1 ;N, GET POINTER TO BAD TRACK TABLE
        MOVEQ.L #0,D4           ;GET SECTOR ADJUSTOR
        TST.W   NBTK$(A0)       ;N, ANY BAD TRACKS?
          BEQ.S @090            ;N, SKIP ALL THIS
*
@070    TST.W   (A1)            ;ANY MORE TO CHECK?
          BEQ.S @080            ;N, JUST ADD IN THESE
        CMP.W   (A1)+,D7        ;Y, IN RANGE?
          BLO.S @080            ;N, JUST ADD IN THESE
        ADDQ.W  #1,D7           ;Y, UP LOGICAL TRACK #
        ADD.L   D6,D4           ;+ SPTK$(A0) = OFFSET IN SECTORS
        BRA.S   @070
*
```

```
@080    ADD.L   D4,D1           ;Y, ADD IN BAD TRACK OFFSET
*
*       ENTER FROM 200-209 PART
*
@090    MOVEQ.L #0,D0           ;CLEAR UPPER WORD
        MOVE.B  DSEL$(A0),D0    ;GET SELECT CODE
        BRA     WFC00           ;EXECUTE IT
*
        PAGE
*
*******************************************************************
*       SUSPEND ON TIMEOUT AND EVENT 119
*         OUT:  .EQ. if not timeout
*         USES: D0,D1
*
SUSPEND MOVE.L  #EVENTO,D0      ;GET 3 SECOND COUNT
        MOVEQ.L #$80,D1         ;GET LOCAL EVENT
        XDEV                    ;DELAY LOCAL EVENT X SECONDS
          BNE.S SUSPEND         ;DIDN'T GET IN, TRY AGAIN
        MOVE.W  #$8000+W$EVNT,D1 ;GET LOCAL/SUSPEND EVENT
        XSUI                   ;SUSPEND ON 119 AND LOCAL EVENT, WAIT...
        CMP.B   D0,D1           ;WAS IT EVENT 119?
        RTS
        PAGE
*
*******************************************************************
*       Check Validity of and then move
*       Winch parms down to low memory
*
*       IN:     D6.W = CNTN$ | DSEL$
*               (A1) = DEFAULT TABLE
*               (A2) = DISK HEADER DATA
*               (A4) = NEXT AVAILABLE PARM AREA
*               (A5) = P$PARM ENTRY
*
*       OUT:    (A1) = PARAM RAM DATA
*               (A4) = NEXT AVAILABLE
*               updates A5
*
DOIT    CMPI.L  #'ME4U',(A2)+   ;IS WNERD INITED?
          BNE.S @010            ;N, USE DEFAULTS
        MOVE.W  (A2),D0         ;IS HEDS <= 0?
          BLE.S @010            ;Y, USE DEFAULTS
        SUBI.W  #16,D0          ;IS HEDS > 16?
          BGT.S @010            ;Y, USE DEFAULTS
        TST.W   SPTK$(A2)       ;PDOS SECTORS PER TRAKS NON-ZERO?
          BEQ.S @010            ;N, USE DEFAULTS
        MOVEA.L A2,A1           ;USE DISK HEADER DATA
*
@010    MOVE.W  NPRT$(A1),D0    ;GET # OF PARTS
        MULU.W  #3,D0           ;= # OF WORDS
        ADD.W   NBTK$(A1),D0    ;+ BAD TRACK ENTRIES
        ADDI.W  #PART$/2+1-1,D0 ;+ 8+1 TERMIN HEADER INFO WORDS
        MOVE.L  A4,(A5)         ;SAVE ADDRESS OF THIS PARM TABLE
*
```

```
@020    MOVE.W  (A1)+,(A4)+
          DBF   D0,@020
        MOVEA.L (A5)+,A1        ;GET PARM ADDRESS
        MOVE.W  D6,CNTN$(A1)    ;SET CONTROLLER & DSEL
        RTS
        PAGE
*****************************************
*       DISK OFF ROUTINE: IF WFC, THEN
*         DESELECT DRIVES AFTER SO MANY SECONDS
*
XDOF    TST.B   P$WFCF          ;IS WFC IN SYSTEM?
          BEQ.S @0002           ;0,1=NO!
        TST.B   STOLOC          ;-1=YES, SELECTED?
          BLE.S @0002           ;N
        SUBQ.B  #1,STOLOC       ;Y, TIME TO TURN OFF?
          BGT.S @0002           ;N
        MOVE.B  #$18,BBASW+SDHD ;Y, SELECT FLOPPY #0 TO DESELECT FLOPPY
*
@0002   RTS
        PAGE
********************************************************************
*
* DRIVER FOR FORCE WFC-1 DISK CONTROLLER
* THIS CONTROLLER IS CALLED FROM THE STANDARD FORCE DISK CONTROLLER
* HEADER.
*
* UPDATE SCHEDULE:
*  25-NOV-86  1.0  M.S.  INITIAL VERSION, DERIVED FROM FORMER FBIOSW
*  09-JUN-87  1.1  M.S.  Init WFC-1 vectors at beginning of INITWFC
*
********************************************************************
*        -- WFC-1 CODE --
*
*         D0.B = SELECT CODE BYTE
*         D1.L = LOGICAL SECTOR #
*         D2.L = READ/WRITE COMMAND (0/1)
*         (A0) = DRIVE PARAMS
*
WFCINIT CLR.W   -(A7)           ;SET .NE. FOR ENTRY FROM XDITW
*
WFC00   ADDQ.L  #2,D2           ;MAKE 2,3
        LSL.L   #4,D2           ;MAKE $20,$30
        LEA.L   BBASW,A1        ;POINT TO WFC BOARD
        DIVU.W  SPTK$(A0),D1    ;D1 = SECT | LOG. TRK
        SWAP    D1
        CMPI.B  #$18,D0         ;IS SELECT CODE 18,1A,1C,1E: FLOPPY?
          BLT.S @010            ;N, WINCHESTER 00,08,10 PLUS ECC BIT
        ADDQ.W  #1,D1           ;N, FLOPPY SECTS START AT 1
*
```

```
@010    MOVE.B  D1,D3             ;SAVE SECTOR #
        CLR.W   D1
        SWAP    D1                ;D1 = LOGICAL TRACK
        DIVU.W  HEDS$(A0),D1      ;D1 = HEAD | CYL
        ROR.W   #8,D1             ;D1 = 0000 | HEAD | CYLL | CYLH
        SWAP    D1                ;   = CYLL | CYLH | 0000 | HEAD
        ROR.W   #8,D1             ;   = CYLL | CYLH | HEAD | 0000
        MOVE.B  D3,D1             ;   = CYLL | CYLH | HEAD | SECT
        ROR.L   #8,D1             ;   = SECT | CYLL | CYLH | HEAD
        OR.B    D0,D1             ;OR IN SELECT CODE
        CLR.B   STOLOC            ;!!! BE SURE DRIVE REMAINS SELECTED
        MOVEP.L D1,SNUM(A1)       ;OUT SNUM,CYL,CYH,SDHD
        MOVE.B  #PREC/4,ERRR(A1) ;OUT PRECOMP TRACK
        MOVE.B  #1,SCNT(A1)       ;OUT 1 SECTOR TRANSFER
*
@020    BTST.B  #IWFC,P$INTF      ;USING WFC INTS?
          BEQ.S @030              ;N, SKIP EVENT RESET
        BCLR.B  #~W$EVNT,W$EVNT/8+EVTB.(A5) ;RESET EVENT 119
*
@030    MOVE.B  D2,COMR(A1)       ;ISSUE R/W COMMAND
        MOVE.W  #256-1,D4         ;GET BYTE COUNT
        MOVEQ.L #20,D3            ;GET TIMEOUT
        SWAP    D3
        CMPI.B  #$30,D2           ;IS THIS A WRITE?
          BNE.S @050              ;N
*
* IT IS WRITE -- CHECK DMA AND TRANSFER DATA
*
@040    BTST.B  #DWFC,P$INTF      ;USE DMA FOR TRANSFER ?
          BEQ.S @049              ;N
        MOVE.L  P$DMAC,A3         ;POINT TO CHIP
        MOVE.B  #$FF,DCSR(A3)     ;RESET DMA
        MOVE.L  A2,DMAR(A3)       ;SET MEMORY ADDRESS
        MOVE.W  #256,DMTC(A3)     ;DO 256 BYTES TRANSFER
        MOVE.B  #$01,DOCR(A3)     ;DO MEM TO DEV ON MAX SPEED
        MOVE.B  #$80,DCCR(A3)     ;GO !!
@044    BTST.B  #7,DCSR(A3)       ;POLL FOR COMPLETION
        BEQ.S   @044              ;WAIT
        BTST.B  #4,DCSR(A3)       ;ERROR ?
        BEQ.S   @050              ;N
        MOVEQ.L #107,D0           ;ERROR 107 = DMA ERROR
        BRA     WFCOUT
*
@049    MOVE.B  (A2)+,DATR(A1)    ;OUTPUT DATA TO CONTROLLER
          DBF   D4,@049           ;LOOP
*
@050    BTST.B  #IWFC,P$INTF      ;USE WFC INTS?
          BEQ.S @060              ;N, JUST POLL BUSY
        BSR     SUSPEND           ;Y, SUSPEND ON EV 119, TIMEOUT?
          BEQ.S @080              ;N, GET STATUS AND DROP INTO FINISH
        BCLR.B  #IWFC,P$INTF      ;Y, DISABLE INTS FOR NEXT TIME
*
@060    MOVEQ.L #102,D0           ;Y, ASSUME TIMEOUT ERROR
*
@070    SUBQ.L  #1,D3             ;TIMEOUT?
          BEQ.S WFCOUT            ;Y, OUT ERROR 102
```

```
*
@080    MOVE.B  COMR(A1),D1     ;GET STATUS REG
          BMI.S @070            ;LOOP, STILL BUSY
        MOVEQ.L #103,D0         ;DONE, ASSUME WRITE FAULT
        CMPI.B  #$20,D2         ;WAS IT A READ?
          BEQ.S @090            ;Y, MOVE DATA IN
        BTST    #5,D1           ;N, WRITE FAULT?
          BNE.S WFCOUT          ;Y, ERROR 103
        BRA.S   @100            ;TO COMMON
*
* IT IS READ -- CHECK DMA AND TRANSFER DATA
*
@090    BTST.B  #DWFC,P$INTF    ;USE DMA FOR TRANSFER ?
          BEQ.S @099            ;N
        MOVE.L  P$DMAC,A3       ;POINT TO CHIP
        MOVE.B  #$FF,DCSR(A3)   ;RESET DMA
        MOVE.L  A2,DMAR(A3)     ;SET MEMORY ADDRESS
        MOVE.W  #256,DMTC(A3)   ;DO 256 BYTES TRANSFER
        MOVE.B  #$81,DOCR(A3)   ;DO DEV TO MEM ON MAX SPEED
        MOVE.B  #$80,DCCR(A3)   ;GO !!
@094    BTST.B  #7,DCSR(A3)     ;POLL FOR COMPLETION
        BEQ.S   @094            ;WAIT
        BTST.B  #4,DCSR(A3)     ;ERROR ?
        BEQ.S   @100            ;N
        MOVEQ.L #107,D0         ;ERROR 107 = DMA ERROR
        BRA.S   WFCOUT
*
@099    MOVE.B  DATR(A1),(A2)+  ;READ FROM BUFFER
          DBF   D4,@099         ;LOOP
*
@100    ADDQ.W  #4,(A7)         ;SET .EQ.
        LSR.B   #1,D1           ;IS ERROR BIT SET?
          BCC.S WFCOUT          ;N, TAKE OK EXIT
        SUBQ.W  #4,(A7)         ;Y, SET .NE.
        MOVE.B  ERRR(A1),D1     ;GET ERROR REG
*
@110    ADDQ.L  #1,D0           ;UP ERROR #
        LSR.B   #1,D1           ;FOUND ERROR BIT?
          BCC.S @110            ;N
*
WFCOUT  MOVE.B  #STO,STOLOC     ;Y, SEND ERROR AND .NE. OR .EQ.
        RTR
        PAGE
```

```
***********************************************************************
*       SYS68K/WFC-1 DISK INIT
*
*       DEFINE: RESTORE ALL DRIVES
*
INITWFC
*
* INIT DMAC
*
        TST.W   P$DMAC          ;DO WE HAVE A DMA ?
         BEQ.S  @000            ;N
        MOVE.L  P$DMAC,A1       ;Y, GET ADDRESS
        MOVE.B  #$FF,DCSR(A1)   ;RESET CSR
        MOVE.B  #2,DDCR(A1)     ;SET BURST, 8-BIT 68000 DEV
        MOVE.B  #4,DSCR(A1)     ;MEM COUNTS UP, DEV DOES NOT
        MOVE.B  #5,DMFC(A1)     ;SUPERVISOR DATA ACCESS
        MOVE.B  #5,DDFC(A1)     ;SUPERVISOR DATA ACCESS
        MOVE.B  #0,DBFC(A1)     ;BFC IS DON'T CARE
        MOVE.L  #BBASW+DATR,DDAR(A1) ;SET DEVICE ADDRESS
*
* NOW DO WFC INIT
*
@000    LEA.L   BBASW,A1        ;POINT TO BOARD
        MOVE.B  #W$EVNT,CIVR(A1) ;SET COMMAND INTERRUPT VECTOR
        MOVE.B  #W$EVNT,DIVR(A1) ;SET DATA INTERRUPT VECTOR, NOT USED
        MOVE.B  #STO,STOLOC     ;SET TIMEOUT
        LEA.L   WSELB(PC),A3    ;POINT TO SELECT BYTE GUYS
        MOVE.W  #$0200,D6       ;CONTROLLER #2 | DSEL 0
*
*       LOOP THRU 3 WINCH DRIVES
*
@010    LEA.L   BBASW,A1        ;POINT TO BOARD
        TST.B   COMR(A1)        ;BUSY?
          BMI.S @010            ;Y, WAIT
        MOVE.B  #10,CYLL(A1)    ;N, TO CYL 10
        MOVE.B  (A3)+,D6        ;GET NEXT SELECT BYTE
        MOVE.B  D6,SDHD(A1)     ;SELECT A DRIVE
*
@020   MOVE.B  (A3)+,COMR(A1)  ;SEND SEEK/RESTORE COMMAND (W/ FLP STEP)
*
@030    TST.B   COMR(A1)        ;BUSY?
          BMI.S @030            ;Y, WAIT
        CMPI.B  #$70,-1(A3)     ;N, WAS LAST COMMAND A WINCH SEEK?
          BEQ.S @020            ;Y, SEND A RESTORE COMMAND
        LEA.L   WFLPARM(PC),A1  ;N, GET DEFAULT FLOPPY PARMS
        MOVEA.L A1,A2           ;GET FAIL FOR DOIT
        CMPI.B  #$1C,D6         ;IS SELECT CODE 1C,1E: FLOPPY?
          BGT.S @050            ;Y, FLOP 3, JUST EXECUTE DOIT
          BLT.S @040            ;N, WINCH, READ HEADER
        EXG     A5,A6           ;Y, FLOP 2, BEGIN USING P$FPARM AREA
        BRA.S   @050            ;GO TO DOIT
*
*       IF WINCHESTER, DO A READ OF SECTOR 0 INTO $2FC
*
```

```
@040    LEA.L   DEFALTW(PC),A0  ;N, GET FAKE WINCH PARAMS
        MOVEQ.L #$00,D0         ;SELECT DRIVE D6
        MOVE.B  D6,D0
        MOVEQ.L #$00,D1         ;SECTOR 0
        MOVEQ.L #0,D2           ;READ COMMAND
        LEA.L   P$BLOCK,A2      ;GET FAKE BUFFER ADDRESS
        BSR     WFCINIT         ;DO A READ SECTOR 0
          BNE.S @070            ;READ ERROR, DO NOT INSTALL DRIVE
        LEA.L   DEFALTW(PC),A1  ;GET DEFAULT WFC PARAMS
        LEA.L   P$BLOCK,A2      ;GET HEADER DATA AREA
*
@050    BSR.L   DOIT            ;MOVE DATA DOWN
        CMPI.B  #$1E,D6         ;IS SELECT CODE 1E: FLOPPY #3
          BNE.S @070            ;N, WINCH OR FLOPPY #2
*
        PAGE
@060    ADDQ.W  #1,PART$(A1)    ;Y, FLOP 3 == DISK #1
        BRA.S   @080            ;DROP OUT ON SECOND FLOPPY
*
@070    CMPA.L  #P$PARM+4*6,A5  ;A PARM TOO FAR?
          BHS.S @080            ;Y
        CMPI.B  #$FF,(A3)       ;DONE ALL DRIVES (3 WINCH 2 FLOP)?
          BNE   @010            ;N, LOOP
*
@080    LEA.L   BBASW,A1        ;POINT TO BOARD
        MOVE.B  #STO,STOLOC     ;SET TIMEOUT
        BSET.B  #IWFC,P$INTF    ;TURN ON WFC INTS
        TST.W   P$DMAC          ;DO WE HAVE A DMA ?
         BEQ.S  @082            ;N
        BSET.B  #DWFC,P$INTF    ;Y, TURN DMA ON
@082    RTS
        PAGE
        PAGE
```

```
*******************************************************************
*       WFC-1 DISK FORMATTER
*
WFCFORM RTS                         ;NOT IMPLEMENTED
*******************************************************************
*
*       DEFAULT WFC-1 HEADER PARTITIONS
*
DEFALTW DC.W    3                   ;WFC HEDS
        DC.W    830                 ;WFC CYLS
        DC.W    32                  ;WFC BPT
        DC.W    256                 ;WFC BPB
        DC.W    0                   ;WFC SHIP
        DC.W    32                  ;WFC SPT
        DC.W    3                   ;WFC # OF PARTS
        DC.W    0                   ;WFC BAD TRACKS
        DC.W    $0200               ;CNTN & DSEL
        DC.W    0                   ;STEP
        DC.W    0                   ;REDU WRT CURR
        DC.W    0                   ;WRT PERCOMP
        DC.W    2
        DC.W    0,1244              ;PART 0
        DC.W    3
        DC.W    1329,2489           ;PART 1
        DC.W    4
        DC.W    1245,1325           ;PART 2
        DC.W    0
*
        PAGE
*************************************
*       FAKE FLOPPY PARTITION TABLE
*
WFLPARM DC.W    2                   ;FLOPPY HEDS
        DC.W    80                  ;CYLS
        DC.W    16                  ;PHYS BPT BLOCKS PER TRACK
        DC.W    256                 ;PHYS BPB BYTES PER BLOCK
        DC.W    0                   ;SHIP
        DC.W    16                  ;# PDOS SPT SECTORS PER TRACK
        DC.W    1                   ;# PARTS
        DC.W    0                   ;# BAD TRKS
        DC.W    $0140               ;CNTN & DSEL
        DC.W    0                   ;STEP
        DC.W    0                   ;REDU WRT CURR
        DC.W    0                   ;WRT PERCOMP
        DC.W    0                   ;DISK #0
        DC.W    0,159               ;PARTITION 0
        DC.W    0
*
        PAGE
```

```
 ****************************************************************
*       WFC-1 INIT TABLE
*
WSELB   DC.B    $00+ECC,$70     ;SEEK WINCH 0
        DC.B    $10             ;RESTORE WINCH 0
        DC.B    $08+ECC,$70     ;SEEK WINCH 1
        DC.B    $10             ;RESTORE WINCH 1
        DC.B    $10+ECC,$70     ;SEEK WINCH 2
        DC.B    $10             ;RESTORE WINCH 2
        DC.B    $1C,$10+FSTEP   ;RESTORE FLOPPY 2
        DC.B    $1E,$10+FSTEP   ;RESTORE FLOPPY 3
        DC.B    $FF             ;TABLE TERMINATOR
        EVEN
*
********************* END OF WFC-1 DRIVER *************************

        END
```

## C.9 Parameters File for VMEPROM

This file sets the parameters for the use of VMEPROM with FORCE controller boards.

```
*       FPARM:SR                    21-OCT-87
*****************************************************************
*       FORCE PARAMETERS FOR VMEPROM
*
        OPT     ARS
        IFUDF   B.VEC :B.VEC  EQU 0
        OFFSET  $0400+B.VEC     ;LOW MEMORY DEFINITION
*
P$BLOCK EQU     $6F0+B.VEC      ;SECTOR BUFFER FOR W$XDIT
*
* BOARDS IN THE SYSTEM ARE COUNTED HERE
*
P$PARMF
P$SASF  DS.B    1               ;400 - RESERVED
P$WFCF  DS.B    1               ;401 - # OF WFC CARDS IN SYSTEM
P$SIOF  DS.B    1               ;402 - # OF SIO-1 CARDS IN SYSTEM
P$ASCF  DS.B    1               ;403 - # OF ASCU BOARDS
P$FPCPF DS.B    1               ;404 - PROCESSOR & 68881 AVAILABLE FLAG
P$SCSIF DS.B    1               ;405 - # OF ISCSI-1 CARDS
P$ISIOF DS.B    1               ;406 - # OF ISIO-1 CARDS
        DS.B    1               ;407 - UNUSED RESERVED
P$MEM   DS.L    1               ;408 - TOP OF MEMORY
        DS.B    1               ;40C - UNUSED, RESERVED
        DS.B    1               ;40D - UNUSED, RESERVED
        DS.B    1               ;40E - UNUSED, RESERVED
P$SWITCH DS.B   1               ;40F - KEEP STANDARD SWITCH SETTINGS
*
* JUMP TABLE TO DISK I/O ROUTINES
*
P$DRW   DS.L    1               ;410 - DISK READ/WRITE HANDLER ADDRESS
P$DFORM DS.L    1               ;414 - DISK FORMAT HANDLER ADDRESS
P$DMAC  DS.L    1               ;418 - DMA CONTROLLER ADDRESS, IF ANY
*
* GENERAL PURPOSE LOCATIONS
*
B$SRAM  DS.L    1               ;41C - HOLDS THE ADDRESS OF SYRAM
P$CPU   DS.W    1               ;420 - CPU BOARD TYPE IS STORED HERE
*
* ISIO HIGH/LOW WATER FLAG DEFINITIONS
* 1 BIT FOR EACH POSSIBLE ISIO-1/2 CHANNEL
*
P$ISINT DS.W    1              ;422 - HIGH/LOW WATER FLAGS FOR ISIO-1/2
*
*
STOLOC  DS.B    1               ;424 - SELECT TIMEOUT (WFC)
P$BDSK  DS.B    1               ;425 - SAVE BOOT DISK #
*
```

```
*       P$INTF FLAG BIT DEFINITIONS:
*
ISAS    EQU     0               ;BIT 0: SASI-1 INTERRUPTS
IWFC    EQU     1               ;BIT 1: WFC-1 INTERRUPTS
DSAS    EQU     2               ;BIT 2: SASI-1 DMA MOVE
DWFC    EQU     3               ;BIT 3: WFC-1 DMA MOVE
ISCSI   EQU     4               ;BIT 4: ISCSI-1 TIME OUT
DSCSI   EQU     5               ;BIT 5: ISCSI-1 DMA MOVE
IISIO   EQU     7               ;BIT 7: ISIO-1/2 HIGH/LOW WATER
*
P$INTF  DS.B    1               ;426 - SUSPEND ON INTERRUPT FLAG
P$LEDB  DS.B    1               ;427 - SPARE (LED BYTE), NOT USED
*
SAVE    DS.L    2               ;428 - REG SAVE AREA
P$UADR  DS.L    1               ;430 - SAVES ADDRESS OF INTERRUPT UART
P$PARM  DS.L    6               ;434 - 2 FLOP, 4 WINC POINTERS
*P$FPARM DS.L   2               ;POINTERS TO FLOPPY DISK PARM TABLE
*P$WPARM DS.L   4               ;POINTERS TO WINCH DISK PARM TABLES
*       ...                     ;VARIABLE ROOM FROM HERE DOWN vvvv
*
*       All of the following will be compiled and
*       loaded by the DISK INIT program into RAM....
*
*       Each Winch has these 8 variables read from its track 00:
*
HEDS$   EQU     0               ;# OF HEADS ON WINCH 0
CYLS$   EQU     2               ;# OF CYLINDERS
BPTK$   EQU     4               ;BLOCKS PER TRACK
BPBK$   EQU     6               ;BYTES PER BLOCK
SHIP$   EQU     8               ;SHIPPING CYLINDER
SPTK$   EQU     10              ;# OF PDOS SECTORS PER TRACK
NPRT$   EQU     12              ;# OF PARTITIONS
NBTK$   EQU     14              ;# OF BAD TRACKS
CNTN$   EQU     16              ;Controller # byte
DSEL$   EQU     17              ;Drive select byte
STEP$   EQU     18              ;Step rate word
REDU$   EQU     20              ;Reduced write current word
WRTP$   EQU     22              ;Write protect word
PART$   EQU     24              ;PARTITIONS START HERE
*
*  DEFINITIONS FOR FORCE I/O CONTROLLER BOARDS WHICH MAY BE IN THE
*  SYSTEM
*
```

```
* 1. SIO-1/SIO-2 BOARD
 *-------------------------------------------------------------------
*
SIOBASE EQU     $FCB00000
*
*       68561 MPCC UART CHIP OFFSETS FOR SIO-1 BOARDS
*
SRSR    EQU     $01             ;RECV STATUS
SRDR    EQU     $03             ;" DATA
SRIVNR  EQU     $05             ;" INTERRUPT VECTOR
SRCR    EQU     $21             ;RECV CONTROL
SRIER   EQU     $25             ;" INT ENABLE
STSR    EQU     $09             ;XMIT STATUS
STDR    EQU     $0B             ;" DATA
STCR    EQU     $29             ;XMIT CONTROL
STIER   EQU     $2D             ;" INT ENABLE
SPSR1   EQU     $19             ;PROTOCOL SELECT 1
SPSR2   EQU     $39             ;PROTOCOL SEL 2
SBRDR1  EQU     $1D             ;BAUD RATE DIVIDER 1
SBRDR2  EQU     $3D             ;BAUD RATE 2
SCCR    EQU     $1F             ;CLOCK CONTROL REG
SECR    EQU     $3F             ;ERROR CONTROL
SSICR   EQU     $31             ;SERIAL CONTROL
SSIER   EQU     $35             ;" INT ENABLE
SSISR   EQU     $11             ;" STATUS
*
*
* 2. WFC-1 CONTROLLER
*------------------------------------------------------------------
*
BBASW   EQU     $FCB01000       ;WFC VME ADDRESS, A24/D16 AREA
WFCBASE EQU     BBASW
*
*       OFFSETS FROM BOARD BASE (BBASW):
*
CIVR    EQU     $00             ;COMPLETE INTERRUPT VECTOR REG
DATR    EQU     $01             ;DATA REG
DIVR    EQU     $02             ;DRQ INTERRUPT VECTOR REG
ERRR    EQU     $03             ;ERROR REG | WRITE PRECOMP
SCNT    EQU     $05             ;SECTOR COUNT
SNUM    EQU     $07             ;SECTOR #
CYLL    EQU     $09             ;CYLINDER LSB
CYLH    EQU     $0B             ;CYLINDER MSB
SDHD    EQU     $0D             ;SIZE/DRIVE/HEAD
COMR    EQU     $0F             ;STATUS | COMMAND REG
*
ECC     EQU     $0              ;=0 CRC, =$80 ECC
PREC    EQU     400             ;WRITE PRECOMPENSATION TRACK
STO     EQU     2               ;MOTOR TIMEOUT IN SECONDS
*
*
```

```
* 3. SASI CONTROLLER
*-------------------------------------------------------------
*
CBASE   EQU     $FC900000+$100  ;SASI CHANNEL 1 BASE ADDRESS
SASIBASE EQU    CBASE
*
*       OFFSETS FROM CHANNEL BASE (CBASE):
*
SCOMF   EQU     0               ;SASI COMMAND FIELD
SSTAT   EQU     $0A             ;SASI STATUS REGISTER
STERM   EQU     $0B             ;SASI TERMINATION
CCONT   EQU     $0C             ;CHANNEL CONTROL
SSENS   EQU     $0D             ;SASI READ SENSE BYTES
CVECT   EQU     $11             ;INTERRUPT COMPLETION VECTOR
EVECT   EQU     $12             ;INTERRUPT ERROR VECTOR
SADDR   EQU     $E0             ;START ADDRESS FOR DATA TRANS
EADDR   EQU     $E4             ;END ADDRESS
AMODC   EQU     $E8             ;ADDRESS MODIFIER CONTROL
TIDEN   EQU     $FF             ;TARGET IDENTIFICATION CONTROL
DATA    EQU     $100            ;DATA ARRAY
*
*
* 4. ASCU-1/2
*---------------------------------------------------------------
*
ASCBASE EQU     $FCB02000
*
*       68561 MPCC UART CHIP OFFSETS FOR ASCU-1/2 BOARD
*
AMPCCB  EQU     ASCBASE+$0
ARSR    EQU     $01             ;RECV STATUS
ARDR    EQU     $03             ;" DATA
ARIVNR  EQU     $05             ;" INTERRUPT VECTOR
ARCR    EQU     $21             ;RECV CONTROL
ARIER   EQU     $25             ;" INT ENABLE
ATSR    EQU     $09             ;XMIT STATUS
ATDR    EQU     $0B             ;" DATA
ATCR    EQU     $29             ;XMIT CONTROL
ATIER   EQU     $2D             ;" INT ENABLE
APSR1   EQU     $19             ;PROTOCOL SELECT 1
APSR2   EQU     $39             ;PROTOCOL SEL 2
ABRDR1  EQU     $1D             ;BAUD RATE DIVIDER 1
ABRDR2  EQU     $3D             ;BAUD RATE 2
ACCR    EQU     $1F             ;CLOCK CONTROL REG
AECR    EQU     $3F             ;ERROR CONTROL
ASICR   EQU     $31             ;SERIAL CONTROL
ASIER   EQU     $35             ;" INT ENABLE
ASISR   EQU     $11             ;" STATUS
*
```

```
*       68230 PI/TIMER CHIP REGISTERS ON ASCU
*
APIT1B  EQU     ASCBASE+$40     ;PI/T #1
APIT2B  EQU     ASCBASE+$80     ;PI/T #2
APGCR   EQU     $01             ;PORT GENERAL CONTROL REG
APSRR   EQU     $03             ;PORT SERVICE REQ REG
APADD   EQU     $05             ;PORT A DATA DIRECTION REG
APBDD   EQU     $07             ;PORT B DATA DIRECTION REG
APCDD   EQU     $09             ;PORT C DATA DIRECTION REG
APIVR   EQU     $0B             ;PORT INT VECTOR REG
APACR   EQU     $0D             ;PORT A CONTROL REG
APBCR   EQU     $0F             ;PORT B CONTROL REG
APADR   EQU     $11             ;PORT A DATA REG
APBDR   EQU     $13             ;PORT B DATA REG
APCDR   EQU     $19             ;PORT C DATA REG
APSR    EQU     $1B             ;PORT STATUS REG
ATMCR   EQU     $21             ;TIMER CONTROL REG
ATIVR   EQU     $23             ;TIMER INT VECTOR REG
ACPR    EQU     $25             ;COUNTER PRELOAD REG (4 BYTES)
ACNTR   EQU     $2D             ;COUNTER REG (4 BYTES)
ATMSR   EQU     $35             ;TIMER STATUS REG
*
*       58167 RTC CHIP OFFSETS FOR ASCU
*
ARTCB   EQU     ASCBASE+$C0
AICR    EQU     $23             ;INTERRUPT CONTROL REG
AISR    EQU     $21             ;INTERRUPT STATUS REG
ASBI    EQU     $2D             ;STANDBY INT CONTROL
ARAM    EQU     $11             ;FIRST RAM COMPARE REG
*
*       BIM CHIP OFFSETS FOR ASCU
*
ABIM1B  EQU     ASCBASE+$100
ABIM2B  EQU     ASCBASE+$110
ABIM3B  EQU     ASCBASE+$120
ABIM4B  EQU     ASCBASE+$130
ABIMCR0 EQU     1               ;CONTROL REGISTER 1
ABIMCR1 EQU     3               ;CONTROL REGISTER 2
ABIMCR2 EQU     5               ;CONTROL REGISTER 3
ABIMCR3 EQU     7               ;CONTROL REGISTER 4
ABIMVR0 EQU     9               ;VECTOR REGISTER 1
ABIMVR1 EQU     $B              ;VECTOR REGISTER 2
ABIMVR2 EQU     $D              ;VECTOR REGISTER 3
ABIMVR3 EQU     $F              ;VECTOR REGISTER 4
*
*
```

```
* 5. ISIO-1/2 SERIAL I/O CONTROLLER
*--------------------------------------------------------------
*
ISIOB   EQU     $FC960000
ISIOB2  EQU     $FC980000
*
*       ISIO-1/2 REGISTER DEFINITIONS
*
ISCMDR  EQU     $8000           ; CMDRAM OFFSET
ISRES   EQU     ISCMDR+15       ; CHECK ADDRESS
GETCHI  EQU     $1010           ; GET CHAR WITH INT
PUTCH   EQU     $0114           ; PUTCHAR WITHOUT INT
SETHS   EQU     $0005           ; SET HANDSHAKE MODE COMMAND
ASINI   EQU     $0006           ; ASYNC INIT
ISABORT EQU     $8100           ; OFFSET TO CHABORT
ISINT   EQU     $1001           ; OFFSET FOR LOCAL INT
ISBCR0  EQU     $0001           ; ISIO BIM CONTROL REGISTERS
ISBCR1  EQU     $0003
ISBCR2  EQU     $0005
ISBCR3  EQU     $0007
ISBVR0  EQU     $0009           ; ISIO BIM VECTOR REGISTERS

ISBVR1  EQU     $000B
ISBVR2  EQU     $000D
ISBVR3  EQU     $000F
*
*
* 6. ISCSI-1  CONTROLLER
*--------------------------------------------------------------
*
ISCSIB  EQU     $FCA00000               ;ISCSI BASE ADDRESS
*
*       FORCE ISCSI-1 Controller Definition
*
ISCCMD1 EQU     ISCSIB+$2100    ;CMDRAM #1
ISCIOB1 EQU     ISCSIB+$2300    ;I/O BUFFER FOR CHANNEL 1
ISCBIM  EQU     ISCSIB          ;OFFSET TO BIM
ISCBC0  EQU     1               ;BIM CONTROL REG
ISCBC1  EQU     3               ;BIM CONTROL REG
ISCBV0  EQU     9               ;BIM VECTOR REG FOR COMPLETION
ISCBV1  EQU     $B              ;BIM VECTOR REG FOR ERROR
*
*
```

```
* 7. DMA CONTROLLER 68450
*-----------------------------------------------------------------
*
* THIS IS NOT A VMEbus DEVICE BUT THE REGISTER DEFINITION IS HERE
* FOR THE DISK I/O MODULE. IF DMA IS TO BE USED AND AVAILABLE IN THE
* SYSTEM, THE ADDRESS OF THE CHIP WITH THE NAME "DMAC" MUST BE DEFINED
* AT LINK TIME BY THE SYSGEN MODULE.
*
*       DMA OFFSETS
*
*       CHANNEL 0
DCSR    EQU     $00                     ; CHANNEL STATUS REGISTER
DCER    EQU     $01                     ; CHANNEL ERROR REGISTER (R)
DDCR    EQU     $04                     ; DEVICE CONTROL REGISTER
DOCR    EQU     $05                     ; OPERATION CONTROL REGISTER
DSCR    EQU     $06                     ; SEQUENCE CONTROL REGISTER
DCCR    EQU     $07                     ; CHANNEL CONTROL REGISTER
DMTC    EQU     $0A                     ; MEMORY TRANSFER COUNT
DMAR    EQU     $0C                     ; MEMORY ADDRESS REGISTER
DDAR    EQU     $14                     ; DEVICE ADDRESS REGISTER
DBTC    EQU     $1A                     ; BASE TRANSFER COUNTER
DBAR    EQU     $1C                     ; BASE ADDRESS REGISTER
DNIR    EQU     $25                     ; NORMAL INTERRUPT VECTOR
DEIV    EQU     $27                     ; ERROR INTERRUPT VECTOR
DMFC    EQU     $29                     ; MEMORY FUNCTION CODES
DCPR    EQU     $2D                     ; CHANNEL PRIORITY REGISTER
DDFC    EQU     $31                     ; DEVICE FUNCTION CODES
DBFC    EQU     $39                     ; BASE FUNCTION CODES
*
*********************** END OF FILE*********************************
```