Thirty Dollars $5

Cromemco®

Cromix-Plus Programmer's

Reference Manual

Cromemco

Cromix-Plus Programmer's

Reference Manual

November 1985

023-5014
Rev. C

CROMEMCO, Inc.
P.O. Box 7400
280 Bernardo Avenue
Mountain View, CA 94039

This manual was produced using a Cromemco System Three computer running under the Cromemco Cromix Operating System. The text was edited with the Cromemco Cromix Screen Editor. The edited text was proofread by the Cromemco SpellMaster Program and formatted by the Cromemco Word Processing System Formatter II. Camera-ready copy was printed on a Cromemco 3355B printer.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

LIST OF APPENDICES

Chapter 1

INTRODUCTION TO CROMIX SYSTEM CALLS

Calls to the Cromix Operating System are formed using a trap #0 followed by a word specifying the system call number. The Cromemco 68000 Macro Assembler (version 01.14 and higher) contains an opcode (JSYS) that forms these two words in the object code. JSYS takes the Cromix system call number as its only operand.

The files **jsysequ.asm, modeequ.asm,** and **bmodeequ.asm** and **tmodeequ.asm** are provided to facilitate programming system calls. These files contain EQUates for all of the system call numbers and mode options so that the calls may be made by name and the numbers need not be remembered. To make use of these files, include them in the source file using the ***include** statement of the assembler.

For example:

```
*include 'jsysequ.asm'
*include 'modeequ.asm'

        move    #stdin,D1       ;standard input channel
        move    #MD_ISPEED,D2   ;input baud rate
        move.l  #S_2400,D3      ;set to 2400 baud
        jsys    #_setmode       ;system call sets the mode
```

All system calls require the specified calling parameters. In addition, some calls return parameters. Parameters are passed in registers as words or long words, depending on the parameter. Values returned are always long words. All registers not specified as containing a returned parameter are preserved through a system call.

The following list summarizes the Cromix Operating System calls.

SUMMARY OF SYSTEM CALLS

_alarm (43h)	sends alarm signal to calling process after # seconds
_boot (56h)	boots new operating system
_caccess (27h)	tests channel access
_cchstat (23h)	changes the status of an open file
_chdup (0Ah)	duplicates a channel
_chkdev (07h)	verifies presence of a device driver in the operating system
_clink (25h)	establishes an additional link to an open file
_close (0Bh)	closes an open file
_create (08h)	creates and opens a file
_cstat (21h)	returns the status of an open file
_delete (06h)	deletes a directory entry
_divd (54h)	divides two unsigned 32 bit integers
_error (1Ch)	displays an error message
_exchg (0Ch)	exchanges filenames of two open files
_exec (4Ch)	executes a program
_exit (46h)	exits from a process
_faccess (26h)	tests file access
_fchstat (22h)	changes the status of a file
_fexec (4Bh)	forks and executes a program
_flink (24h)	establishes a link to a file
_fork (47h)	forks a user program
_fshell (48h)	forks a Shell process
_fstat (20h)	returns the status of a file
_getdate (30h)	returns the date
_getdir (02h)	returns the current directory pathname
_getgroup (36h)	returns the group id
_getmode (12h)	returns the characteristics of a character device
_getpos (10h)	returns a file pointer
_getprior (38h)	returns the priority of the calling process
_getproc (3Ah)	returns the PID of the calling process
_gettime (32h)	returns the time
_getuser (34h)	returns the user id of the current process
_indirect (51h)	executes system call identified by number in the D0 register
_kill (41h)	sends a signal to a process
_lock (3Eh)	assists in implementing inter-process communications
_makdev (00h)	creates a new name for a device
_makdir (01h)	creates a new directory
_memory (50h)	allocates or deallocates memory
_mount (04h)	enables access to a file system
_mult (53h)	multiplies two unsigned 32-bit integers
_open (09h)	opens a file for access
_pause (44h)	suspends execution and waits for a signal
_pipe (0Eh)	creates a pipe
_printf (1Bh)	generates formatted output
_ptrace (4Eh)	runs a process debugger
_rdbyte (16h)	reads a byte
_rdline (18h)	reads a line
_rdseq (14h)	reads the specified number of bytes
_setdate (31h)	changes the date

_setdir (03h)	changes the current directory
_setgroup (37h)	changes the group id
_setmode (13h)	changes the characteristics of a character device
_setpos (11h)	changes the position of the file pointer
_setprior (39h)	returns the priority of the calling process
_settime (32h)	changes the time
_setuser (35h)	changes the user id
_shell (49h)	initiates a Shell process
_signal (40h)	sets up a process to receive a signal
_sleep (42h)	puts a process to sleep
_trunc (0Dh)	truncates an open file
_uchstat (29h)	changes the status of a process
_unlock (3Fh)	is used to unlock a locking sequence
_unmount (05h)	disables access to a file system
_update (52h)	updates all open files
_ustat (28h)	returns the status of a process
_version (55h)	returns the operating system version number
_wait (45h)	waits for the termination of a child process
_wrbyte (17h)	writes a byte
_wrline (19h)	writes a line
_wrseq (15h)	writes sequential bytes

SIGNALS

A signal carries messages between processes. There are eight types of signals that can effect eight different responses from a process. The programmer can choose any one of three responses to each of seven of the eight types of signals. The **SIGKILL** signal in all cases, causes a process to be aborted.

Responses to a Signal

When a process receives a signal, the signal can be handled in one of three ways.

1. **Ignore the signal.**
   The process continues as though no signal had been received.

2. **Abort the process.**
   The operating system terminates the process. This is equivalent to execution of the **_exit** system call.

3. **Transfer control.**
   A user program may establish a location to which control may be transferred for each type of signal received.

After a signal has been received, the **_signal** system call must be executed again in order to be able to receive the next signal.

Types of Signals

The eight types of signals are enumerated below.

1. **sigabort**
   This is the abort signal generated by a CONTROL-C typed at the terminal. The mode of the terminal must be set to allow CONTROL-C to function (abortenable).

2. **siguser**
   This is the user signal generated by a character typed at the terminal. The character that generates this signal is determined and enabled by mode (sigcharacter and sigenable).

3. **sigkill**
   This is the kill signal. It cannot be ignored or redirected by the user program. The kill signal causes the operating system to abort the process immediately. The kill signal can only be sent to a process by the initiator of the process or a privileged user.

4. **sigterm**
   This is the terminate signal. It is the default type of signal for the Kill command of the Shell.

5. **sigalarm**
   This is the alarm signal. It is sent by the operating system following an **_alarm** system call.

6. **sigpipe**
   This is the pipe signal. It is sent by the operating system when a pipe is not being used properly.

7. **sighangup**
   This is a signal sent by the mtty device when the phone hangs up, if the HUPENABLE mode is set.

8. reserved for future use.

Sources of Signals

Signals may be sent to a process by a user-typed character, the Kill command, the **_kill** system call, the **_alarm** system call, or by a driver.

Reception of Signals

A process may be set up to receive and process a signal by the **_signal** system call. If the signal is not ignored and the process has an unsatisfied request for input or output from a character device such as a terminal or printer, the input or output request is canceled.

A child process may be set up by its parent process to ignore or be aborted by a signal when the parent initiates the child through the **_fexec** or **_fshell** system call.

Reactions to signals are determined by the values of the D1 and D2 registers:

Bit S-1 in D1	Bit S-1 in D2	Child's reaction to signal S
0	x	same as the parent process
1	0	abort
1	1	ignore

If the child is set up to inherit the parent's reactions and the parent process is set up to trap the signal, the child process can still be aborted by the signal. This is because the child process cannot inherit the parent's trap routine.

The **_signal** system call sets up a process to receive a signal. The type of signal to be received is loaded in the D2 register. The execution address is loaded in the A0 register. This is the address to which control is passed once the signal is received. The previous execution address is returned in the A0 register.

Processes initiated by the Shell are set up to inherit reactions to all signals from the parent process, except for the sigabort, siguser, and sigterm signals (these are handled separately).

A process which is run as a detached job by the Shell (through the use of the symbol & on a command line) is set up by the Shell to ignore sigabort and siguser and to be aborted by sigterm. A process which runs in the foreground (not detached) is set up by the Shell to react the same way as the parent process (except for interactive Shell processes, which are always set up to ignore those three signals). These features allow the user to abort the current process by entering CONTROL-C, while not affecting detached processes and allow implementation of the Shell command **kill 0**. Additional precaution is taken that the parent process will not be aborted while the child process is still active.

The **_kill** system call sends signals to processes. The identification number of the process to which the signal goes is loaded in the D3 register. The number of the signal type sent is loaded in the D2 register. A user may only send a signal to a process which that user initiated. Only a privileged user may send signals to processes initiated by other users. When a signal is sent to process 0, that signal is sent to all processes initiated from the terminal where the user who invoked the call logged on. If a privileged user sends SIGUSER to process 1, system shutdown is initiated. When SIGABORT is sent to process 1, the Cromix system consults the **/etc/ttys** file to log on any terminals that have been added and log off any deleted terminals.

If the user program decides to catch a signal, the signal routine must be written in assembly language for the following reason. Signal routines are treated as interrupts in the sense that they must preserve all the registers, including the Condition Code Register (CCR). In a higher level language this requirement cannot be met. Of course, it is possible to write only the interface in an assembly language. The interface will save all required registers, possibly set up some other registers, and then call a higher level language function to do the real job.

The Use of Signals in Application Programs

The **_signal** system call is commonly used to catch or ignore CONTROL-C (sigabort) or other signals. A typical example is a text editor. An editor must catch or ignore CONTROL-C, entered by the user, to avoid possible disaster when the editor is terminated in the midst of file modification. By loading the A0 register with 1 before any **_signal** system call is made, the programmer causes the signal to be ignored. To cause the system to perform a specific function on receiving a CONTROL-C, the programmer loads the A0 register with an address to which execution passes when the signal is received.

Immediately after a signal is received, the process is automatically set up to ignore further signals. If the process is to receive and handle any further signals, the **_signal** system call must be repeated.

If the A0 register is loaded with 00000000 before a **_signal** system call is made, execution of the process will be aborted when a signal of the type specified in the D2 register is received. If the **_signal** system call is not sent, the process is aborted when any signal is received.

Signals have many uses, but they also have limitations. Signals are designed to terminate processes or wake them up. Signals are not interrupts. Signals can be ignored, but not disabled. Mutual exclusion cannot be easily achieved with signals. If an application requires that a process receive and process several signals per second from one or more processes, difficulties with stack overflow are likely to arise.

The program fragment in the following example catches the sigabort signal sent by a CONTROL-C entered on the keyboard. This might be useful in a program such as an editor in which program termination by a CONTROL-C could cause data loss.

```
; Program fragment demonstrating the use of the Signal system call
;  to catch a SIGABORT (^C) signal. The program can only be killed
;  from another terminal.
;
; (Must be assembled with -68010 option)
;

*include '/equ/jsysequ.asm'

start:  bsr     sigsetup
again:  bra     again

sigsetup:
        lea     abort_vector,A0    ; Address of routine to handle CONTROL-C
        move    #sigabort,D2       ; Load signal-type to catch
        jsys    #_signal           ; Make Cromix signal system call
        bcs     error              ; If error then jump to error routine
        rts                        ; Else return

; ABORT_VECTOR - Location where control is to pass after receiving a
; sigabort signal.
abort_vector:
        move    CCR,-(SP)
        push.l  D0-D2/A0
        lea     message,A0         ; Load address of message string
        move    #stdout,D1         ; Standard output channel
        jsys    #_printf           ; Print message on console
        bcs     error              ; If error jump to error routine
        bsr     sigsetup           ; Set up trap routine again
        pop.l   D0-D2/A0
        move    (SP)+,CCR
        rts

error:
        move    #stderr,D1         ; Channel for error messages
        jsys    #_error            ; Call Cromix to write the error message
        move    #-1,D3             ; Set error code
        jsys    #_exit             ; Exit to operating system

message:
        dc.b    'I do not want to be dead\n\0'

        end     start
```

Signals and Forking a New Process

Whenever the user forks a new process which does not fiddle with signals, the forking can be quite simple: the child process should simply inherit signal treatment from the parent process. In more complex cases, there is one pitfall that has to be avoided. It should never happen that the parent process gets killed while the child process is still alive. If this happens, the grandparent process, which is most likely an interactive Shell, will wake up and fight his grandchild process over the characters being input from the terminal. Under such circumstances, the user can never tell which process is going to pick up characters typed on the terminal.

If the child process can set up its own response to signals (it is certainly able to do so if it is an interactive Shell) the parent process must be much more careful. A simple solution is for the parent process, before forking the child process, to set itself up to ignore all signals, storing the old reactions. After the child terminates, the parent process can restore the reactions to their original state. This solution is not always satisfactory: if the user presses CONTROL-C while the child process is running, the parent process will ignore it, though the user might have intended to kill both processes.

A reasonably complete solution can be described as follows:

1. Set up to ignore all signals, storing the old reactions.

2. Inspect the old reactions. If an old reaction was to ignore the signal, keep it that way. If an old reaction was to abort or to trap the signal, a new trap is to be installed. The new trap function (one for each signal) should only note the fact that it was called.

3. Fork a new process with whatever signal reactions are desired, and wait until it terminates.

4. Restore the old signal reactions.

5. If a signal was received in the interim, send the same signal to yourself, thereby causing the same effect (except for the fact that it is postponed).

This description is still not complete, as it does not say what should happen if more than one signal is received in the meantime. This can be handled by the new trap functions and by the processing after the child process terminates. New trap functions can simply set a bit in a word initialized to zero and not establish the trap again. If so, at the end we have a list of signals received while the child was running. The program can now decide which signal to send to itself and in what order (if there is more than one).

The Alarm System Call

After a specified number of seconds, the **_alarm** system call sends an alarm signal (SIGALARM) to the process that made the system call. The **_signal** system call is first used to set up the process for receiving the SIGALARM signal. A typical use of **_alarm** provides a time out feature for a program. If a process must be prevented from hanging on an input request indefinitely, the process first makes the **_alarm** system call. The **_alarm** system call specifies the number of seconds to wait after making the request for input.

The Pause System Call

The **_pause** system call is frequently used in conjunction with the **_alarm** system call. The **_pause** call suspends execution of the calling process and waits for a signal. The **_pause** call does not require the **_signal** system call to set up the process to receive the signal. It is ideal for putting a process to sleep until another process signals it to continue. The **_pause** and **_alarm** calls can be used together to put a process to sleep for a specified number of seconds. For example:

```
sleep10:move.l  #10,D3          ; Send Alarm in 10 seconds
        jsys    #_alarm         ; Call Cromix
        bcs     error           ; If error then jump to error routine
        jsys    #_pause         ; Wait for a signal
        bcs     error           ; If error then jump to error routine
```

The Sleep System Call

The equivalent of the routine above can be achieved with one system call, **_sleep**. The **_sleep** call stops execution of a process for a specified number of seconds. The result shown above can be accomplished as follows using **_sleep:**

```
sleep10:move.l  #10,D3          ; Set to go to sleep for 10 seconds
        jsys    #_sleep         ; Call Cromix
        bcs     error           ; If error then jump to error routine
```

Locks

The **_lock** system call assists in implementing file locks, and allows the operating system to absorb part of the overhead involved in the procedure. No locks are imposed by the operating system; this is done by the application program. The **_lock** and **_unlock** calls merely make and delete entries in a table residing in system memory.

The **_lock** system call enters a string in the lock table. This string is the unique identifier of a record in a file. The string is hereinafter referred to as the **lock sequence.** Should another process make a **_lock** system call using a lock sequence currently in the lock table, the Cromix Operating System does one of two things. It either puts the process to sleep until the entry is removed, or it returns with an error code set. An entry is removed from the table when the process that made the original **_lock** system call reverses it with an **_unlock** system call, followed by the same lock sequence. Any process put to sleep while attempting to lock that sequence is awakened and allowed to make an entry in the table.

The problem of record level lock is resolved by preceding any read or write to a file or record with a **_lock** system call. This achieves mutual exclusion for records and avoids the undesirable effects of having multiple processes reading and writing the same file or record.

The other considerations associated with the **_lock** system call are the type of lock to be made and the character string to be used as the lock sequence.

Shared and Unshared Locks

A shared lock allows other processes access to the lock. Shared locks are typically used when a file is being read. A shared lock does not prevent other processes from entering the file, so that a process that is reading a record does not prevent another process from reading the file. A process attempting to establish an unshared lock when a shared lock has been granted either is put to sleep or receives an error.

Unshared locks are typically used during a write to a file, since they prevent any other process from getting access to the lock sequence. If a process has an unshared lock, any other process attempting to lock the same sequence either is put to sleep or receives an error.

Conditional and Unconditional Locks

Locks can be made conditionally or unconditionally. A conditional lock returns with an error code set if the sequence specified cannot be locked. An unconditional lock puts the calling process to sleep if the sequence is currently locked. The process put to sleep awakens when the process that originally issued the **_lock** call issues an **_unlock** call.

The programmer must decide whether to use a conditional or unconditional lock. For many applications, putting a process to sleep for a brief period because another process has locked a file or record does no harm. In other cases, such a maneuver may suspend execution of a program indefinitely while waiting for some process to unlock a file or record. In this case, a conditional lock may be used to print an error code informing the user that the record or file is in use. An ideal strategy might employ both techniques, or use the **_alarm** system call to prevent indefinite postponement of file access.

Locking Schemes

If more than one program is relying on the **_lock** system call, a mutually agreed upon scheme must be devised so that all programs use the same identifier to reference records in a file. This identifier is the locking sequence and may contain from one to 16 bytes. An example of a locking sequence is the first 8 bytes of the filename followed by the number of the record to be locked. This scheme works as long as no two files simultaneously in use have names beginning with the same eight characters, and as long as two different processes do not access the same file through two links having different names.

A more elaborate locking scheme uses the file device and inode numbers. The combination of device and inode numbers is a unique file identifier. The number of the device on which a file resides can be obtained by using the **_fstat** system call. The locking sequence could be composed of a device number followed by an inode number and a record number.

If the number of available locks is exceeded, the operating system returns from a **_lock** system call with an error message. This message merely indicates there is no room left in the lock table.

A **?deadlock** error is returned if the operating system detects a deadlock condition.

All locks installed by a process are automatically unlocked when the process is terminated.

Sample Implementations of Locks

The uses of record locks are best shown through illustration. Consider an inventory management system on a multi-user Cromix system at a music store. If salesperson A sells a guitar and wishes to decrement the inventory record, the program would enter a section of code designed to perform the following functions:

1. Request record number to read.

2. Lock the record with a shared, unconditional lock.

3. Read the record.

4. Unlock the record.

The program might then inform the salesperson that three guitars are in stock. The salesperson rings up the sale, decrements the count of guitars in stock to two, and writes the record to the database using an unshared conditional lock during the write. Difficulties arise if another salesperson, B, also sells a guitar at the same time. B might read the record at the same time as A, decrement the inventory, and write the file out to the database. The record shows that two guitars are in stock, when in fact, there is now only one.

There are several possible solutions to the problem. The simplest is to make an unshared lock at the time of the original read and perform the unlock only after the record had been written out. The problem with this scheme is the potential for barring another user from access to the record for a long time.

A more adequate solution to the problem is to let the system resolve possible conflicts. All user reads are preceded by a shared lock, which permits simultaneous access of the record by other users. When the modified record is to be written out, the system checks to see if the record has been modified in the interim period. If it has not been changed, it is written out. If it has been changed, the value of the record must be recalculated.

CROMIX SYSTEM CALL ERRORS

If the Cromix Operating System cannot complete a system call in the normal manner, for example, when a program tries to open a file which does not exist, an error condition is generated. This error condition is reflected by the state of the carry flag which is set or reset by the operating system when returning from a system call. If the carry flag is reset (=0), the system call completed its task successfully. If the carry flag is set (=1), the system call ended abnormally and the error type is returned in the D0 register. The D0 register may then be compared with a value from the error definition table in the **jsysequ.asm** file for user exception processing. The carry flag should be checked after every system call except for the **_exit** and **_error** calls. The **_exit** call does not return, and if the **_error** call returns an error, it is possible to generate an endless loop - an error routine which generates an error and then jumps to itself again.

If the **_error** system call is executed after a system call that generated an error, (carry set), an ASCII message equivalent to the error type is sent to the channel specified by the D1 register. (See the **_error** system call.)

The following example attempts to open a file that does not exist. When the file is not found, the program jumps to a create routine. Any other errors fall through to the **_error** system call, which displays the error on the console and then exits to the operating system.

```
               lea      pathname,A0       ; Pathname of file
               move     #OP_RDWR,D2       ; Read  and write access
               jsys     #_open            ; Open the file
               bcc      open_ok           ; No errors, go to open_ok
               cmp      #_notexist,D0     ; If file not found
               beq      create_it         ;   go to create routine
                                          ; else let system process the error
               move     #stderr,D1        ;   stderr channel for console
               jsys     #_error           ;   send the error to the console
               move     #-1,D3            ; Value returned to shell
               jsys     #_exit            ; Exit to operating system

     open_ok:move       #0,D3             ; Value returned to shell
               jsys     #_exit            ; Dummy open routine,
                                          ;   exit to operating system

     path_name:         dc.b     '/usr/non_existent_file\0'
```

Error Conditions

If the Cromix Operating System cannot complete a system call in the normal manner, an error is generated. The operating system flags an error condition by setting the carry bit in the flag register (the carry flag). A normal return from a system call is indicated by a reset carry flag.

If an error has occurred (carry flag is set or is equal to one), the D0 register contains the error code. The type of error that was returned may be established by comparing the D0 register with the following list of error codes. Each error code is preceded by the error number.

29 **?arglist** The argument list that was provided is incorrect.

28 **?argtable** The argument table is exhausted.

15 **?badcall** The system call that was specified is illegal.

1 **?badchan** An invalid channel number was specified. The operating system must be called with a channel number assigned at the time a file was opened or created.

54 **?badformat** The format of the file is bad.

42 **?badfree** A block is out of range in the free list.

43 **?badinum** The inode number is out of range.

52 **?badio** The input or output is bad.

8 **?badname** The filename that was specified does not conform to proper filename syntax. The name is too long or contains illegal characters.

47 **?badpipe** An attempt was made to write to a broken pipe.

34 **?badvalue** The specified value was out of range.

56 **?cdossim** The CDOS simulator is required.

40 **?chnaccess** An attempt has been made to access a channel which the current user may not access.

57 **?corrupt** The system image has been corrupted.

49 **?deadlock** A possible deadlock condition has been detected.

36 **?devopen** a device open error has occurred.

31 **?difdev** There is a cross device link. File references cannot exist across disks.

9 **?diraccess** An attempt has been made to access a directory which the current user may not access. Make sure the pathname does not include any directories with privileged access.

37 **?diruse** An attempt was made to delete a directory that was in use. All files must be deleted from a directory before it may be deleted.

4 **?endfile** An end of file condition exists on the file being processed. There is no data in the file beyond (in a forward direction from) the current file position.

11 **?exists** An attempt has been made to create a file that already exists.

10 **?filaccess** An attempt has been made to open a file to which the current user has no access.

16 **?filsize** The size of the file is too big.

6 **?filtable** The file table has been exhausted.

38 **?filuse** The requested file is an exclusive access file and was in use.

22 **?fsbusy** The requested file system was busy.

14 **?inotable** The inode table is exhausted.

5 **?ioerror** A physical data transmission error has occurred.

19 **?isdir** The specified pathname is that of a directory.

50 **?lcktable** The lock table is exhausted.

49 **?locked** The specified sequence is already locked.

17 **?mnttable** The mount table is exhausted.

25 **?nochild** There is no child process.

32 **?nodevice** There is no device driver for the referenced device.

13 **?noinode** No inodes are left.

39 **?nomatch** There is no match on the specified ambiguous pathname.

26 **?nomemory** There is not enough memory.

45 **?noproc** The process does not exist.

12 **?nospace** An attempt has been made to write to a full disk.

53 **?not68000** 68000 programs cannot be run under Z80.

21 **?notblk** The specified device is not a block special device.

35 **?notconn** The requested I/O device was not connected to the system.

41 **?notcromix** The specified disk is not compatible with the Cromix Operating System.

18 **?notdir** The specified pathname was not that of a directory.

7 **?notexist** The specified file does not exist. Make sure that the pathname properly identifies the desired file.

24 **?notmount** The specified device was not mounted prior to the call.

3 **?notopen** The specified channel has not been opened or was closed prior to the system call. A file must be opened (using the **_open** or **_create** call) prior to being used for I/O.

23 **?notordin** The requested file is not an ordinary file.

30 **?numlinks** This operation would have created too many links to the specified file or device.

27 **?ovflo** An overflow occurred during a divide operation.

20 **?priv** An attempt was made to invoke a privileged system call by other than a privileged user.

44 **?readonly** The device is mounted for read access only.

55	**?runaway**	A runaway program has been aborted.
46	**?signal**	The system call was aborted.
51	**?tapeio**	There has been a tape I/O error.
2	**?toomany**	All possible channels are already open.
33	**?usrtable**	The user process table is exhausted.

Chapter 2

CROMIX-PLUS SYSTEM CALL DESCRIPTIONS

system call: _alarm
number: 43h
purpose: This call sends an alarm signal to the calling process after the specified number of seconds.

user access: all users

summary:

```
move.l  <number of seconds>, D3
jsys    #_alarm
```

calling parameters: D3.L — The D3.L register contains either the number of seconds before an alarm signal is sent to the current process or a zero to cancel the previous alarm.

return parameters: none

possible errors: none

The _alarm call sends an alarm signal to the current process after the specified number of seconds has elapsed. If the D3.L register is loaded with 0 and the _alarm call is executed after an alarm has been set up, the previous alarm is canceled.

system call:	_boot	
number:	56h	
purpose:	This call boots a new operating system.	
user access:	privileged user	
summary:	lea	<address of new system>, A0
	move.l	<size>, D1
	jsys	#_boot
calling parameters:	A0	The A0 register points to the first word of the code for the new operating system.
	D1.L	The D1 register contains the length of the new operating system (in bytes)
return parameters:	none	
possible errors:	?priv	

The _boot system call saves the given 68000 code and performs a shutdown. After shutdown, instead of going into an infinite loop, jsys _boot will move supplied code at address 000000h, load

D1.L = Size of code in bytes
D2.L = Current root device

and then simulate the reset function (i.e., load SP from 000000h, PC from 000004h, and return).

system call: _caccess
number: 27h
purpose: This call tests channel access.

user access: all users

summary:

```
move    <channel>, D1
move    <access bits>, D2
jsys    #_caccess
```

calling parameters:

D1 The D1 register contains the number of the channel whose access is to be tested.

D2 The D2 register contains the access bits to be tested. These bits can be ORed together to test for various combinations of access privileges. These bits may be represented by:

AC_READ	read
AC_EXEC	execute
AC_WRIT	write
AC_APND	append

return parameters:

The carry flag is reset (=0) if the file represented by the channel, is allowed to be open for the specified access.

The carry flag is set and the D0 register contains the error code _fileaccess if the file cannot be open for the specified access.

Note: The _caccess call does not test how the file is open. It tests how the file could be opened.

possible errors:

?fileaccess
?notopen

The _caccess call tests the access privileges of an open channel.

system call: _cchstat
number: 23h
purpose: This call changes the status of an open file.

user access: see table

summary:

```
move    <channel>, D1
move    <status type>, D2
move    <new value>, D3
move    <access mask>, D4     (only for access)
lea     <buffer>, A1          (only for times)
jsys    #_cchstat
```

calling parameters:

D1 The D1 register contains the channel number associated with the open file.

D2 The D2 register contains the status type to be changed.

For access privilege changes:

D3 The D3 register contains the new value of the specified status type.

D4 The D4 register contains the mask of the status bits to be changed:

AC_READ	read
AC_EXEC	execute
AC_WRIT	write
AC_APND	append

For time changing calls:

A1 The register A1 points to a 6-byte buffer contain the new time (ymdhms).

For other status changes:

D3 The D3 register contains the new value.

return parameters: none

possible errors:
?filaccess
?priv
?notopen

The _cchstat call changes the access privileges associated with a file, the times associated with a file, the owner id of a file, or the group id of a file.

The file must be open; the channel number is used to identify the file.

Table of Cchstat Calls

* Who	D2 Register	Status Type	Location of New Information
p	ST_OWNER	owner id	D3 = new value
p	ST_GROUP	group id	D3 = new value
p&o	ST_AOWNER	access owner	D3 = value, D4 = mask
p&o	ST_AGROUP	access group	D3 = value, D4 = mask
p&o	ST_AOTHER	access public	D3 = value, D4 = mask
p	ST_TCREATE	time created	A1 -> 6 byte buffer
p	ST_TMODIFY	time last modified	A1 -> 6 byte buffer
p	ST_TACCESS	time last accessed	A1 -> 6 byte buffer
p	ST_TDUMPED	time last dumped	A1 -> 6 byte buffer

*
p = privileged user
o = owner

system call: _chdup
number: 0Ah
purpose: This call duplicates a channel.

user access: all users

summary:

```
move      <existing channel>, D1
jsys      #_chdup
move.l    D2, <duplicate channel>
```

calling parameters: D1 — The D1 register contains the existing channel number.

return parameters: D2.L — The D2.L register contains the duplicate channel number assigned by the system.

possible errors:
?notopen
?toomany

The _chdup call duplicates a channel and may be used for channel number manipulation. Please refer to the _pipe system call for additional information.

system call: _chkdev
number: 07h
purpose: This call verifies the presence of a specified device driver in the operating system.

user access: all users

summary:

```
move    <type of device>, D2
move    <major device number>, D3
move    <minor device number>, D4
jsys    #_chkdev
```

calling parameters:

D2 The D2 register indicates the type of device:

IS_BLOCK	block device
IS_CHAR	character device

D3 The D3 register contains the major device number.

D4 The D4 register contains the minor device number.

return parameters: none

possible errors: ?nodevice

The _chkdev call verifies the presence of a device driver. If the device driver is present in the operating system, the _chkdev call returns without an error (carry flag clear). If the device driver is not present, the carry flag is set by the call and an error is returned.

system call: _clink
number: 25h
purpose: This call establishes an additional link to an open file.

user access: all users

summary:

```
move    <channel>, D1
lea     <new pathname>, A1
jsys    #_clink
```

calling parameters:

D1 The D1 register contains the channel number of the open file.

A1 The A1 register points to the file pathname to be established (i.e., the new pathname). The pathname must be terminated by a null character.

return parameters: none

possible errors:

?badname
?isdir
?numlinks
?diraccess
?exists
?notopen

The _clink call establishes a link from the file open on the specified channel to the new pathname. The new file pathname must not exist before the _clink call is made.

system call: _close
number: 0Bh
purpose: This call closes an open file.

user access: all users

summary:

```
move    <channel>, D1
jsys    #_close
```

calling parameters: D1 The D1 register contains the channel number of the open file.

return parameters: none

possible errors: ?notopen

The _close call flushes all buffers associated with the specified channel number and disassociates the channel number from the file to which it was assigned.

system call: _create
number: 08h
purpose: This call creates and opens a file.

user access: all users

summary:

```
lea      <pathname>, A0
move     <access mode>, D2
move     <exclusive mask>, D3
jsys     #_create
move.l   D1, <channel>
```

calling parameters:

A0 The A0 register points to a buffer containing the pathname of the file to be created and opened. The pathname must be terminated by a null character.

D2 The D2 register contains the access mode value for opening the file. The following labels represent the values of the D2 register required to establish the desired access mode. The specified access mode is applicable to the current process.

Nonexclusive access values:

OP_READ	read only
OP_WRITE	write only
OP_RDWR	read/write
OP_APPEND	append

Exclusive access values:

OP_XREAD	read only
OP_XWRITE	write only
OP_XRDWR	read/write
OP_XAPPEND	append

If exclusive access is desired, one of the four exclusive access values listed above must be loaded into the D2 register. This, in conjunction with the desired exclusion bit(s) in the D3 register, denies other users access.

The following values may be ORed with the desired access value (see above) to select the truncate option or conditional option.

Truncate flag:

OP_TRUNCF	delete existing data

Conditional flag:

OP_CONDF	return error if file exists

D3 The D3 register contains the mask for exclusive access. It is inspected only if the D2 register indicates exclusive access. Each of the specified bits must be set to prevent the file from being opened by another process for the specified access. (For example, ^OP_READ indicates that no other process may open the file with the read access. This does not exclude another process from opening the file for read/write access. To exclude all reads, ^OP_READ and ^OP_RDWR must be ORed together.) The following values may be ORed together to set more than one bit.

Exclusive access bits:

^OP_READ	exclude read
^OP_WRITE	exclude write
^OP_RDWR	exclude read/write
^OP_APPEND	exclude append

return parameters: D1.L The D1.L register contains the channel number that the system assigned to the file.

possible errors:

?filtable
?badname
?diraccess
?isdir

The _create call creates a file with the specified pathname.

If the file does not exist at the time of the system call, it is created and opened with the requested access.

If the file does exist and the conditional flag is set, an error is returned. If the file does exist and the conditional flag is reset (=0), the file is opened.

If the file exists and is opened (as specified by the conditional flag), the existing data is kept if the truncate flag is reset. The data is discarded (the file is truncated) if the truncate flag is set. A file may only be truncated if the user has write access to the file.

The channel number that the Cromix Operating System returns is used for subsequent access to the file.

The file created has default access privileges. In a standard system, these are read and execute for group and public, and read, execute, write, and append for the owner.

system call: _cstat
number: 21h
purpose: This call returns the status of an open file.

user access: all users

summary:

```
move    <channel>, D1
move    <status type>, D2
lea     <buffer>, A1       (if necessary)
jsys    #_cstat
<depends on status type>
```

calling parameters:

D1 The D1 register contains the channel number associated with the open file.

D2 The D2 register contains the request to the system for the desired information.

A1 The register A1 may point to a 6-byte or 128-byte buffer. Refer to the table.

return parameters: See table

possible errors: ?notopen

The _cstat call returns channel status information. The file must be open; the channel number is used to identify the file. Please refer to the following table of _cstat calls.

Table of Cstat Calls

D2 Register	Information Returned	Location of the Information Returned
ST_ALL	all of inode	A1 -> 128-byte buffer
ST_OWNER	owner id	D3.L
ST_GROUP	group id	D3.L
ST_AOWNER	access owner	D3.L
ST_AGROUP	access group	D3.L
ST_AOTHER	access public	D3.L
ST_FTYPE	file type	D3.L = IS_ORDIN IS_DIRECT IS_CHAR IS_BLOCK IS_PIPE
ST_SIZE	file size	D3.L
ST_NLINKS	number of links	D3.L
ST_INUM	inode number	D3.L
ST_TCREATE	time created	A1 -> 6-byte buffer
ST_TMODIFY	time last modified	A1 -> 6-byte buffer
ST_TACCESS	time last accessed	A1 -> 6-byte buffer
ST_TDUMPED	time last dumped	A1 -> 6-byte buffer

ST_DEVNO	device number	D3.L = major device #
		D4.L = minor device #

ST_DEVICE	device number	D3.L = major device #
		D4.L = minor device #

ST_PDEVNO	device number	D3.L = major device #
		D4.L = minor device #

ST_DEVNO returns the device numbers of the device specified in a device file. If the specified file is not a device file, ST_DEVNO returns zeroes. ST_DEVICE returns the device numbers of the device on which the specified file resides.

ST_PDEVNO returns the same value as ST_DEVNO except: for block device number zero the device number of the root device is returned; for character device number zero the device number of the user's terminal is returned.

system call:	_delete	
number:	06h	
purpose:	This call deletes a directory entry.	
user access:	all users	
summary:	lea	<pathname>, A0
	jsys	#_delete
calling parameters:	A0	The A0 register points to a buffer containing the path name of the directory or file to be deleted. The pathname must be terminated by a null character.
return parameters:	none	
possible errors:	?diraccess ?notexist ?badname	

The _delete call attempts to remove the specified directory entry. If the removed directory entry is the last link to the file, the file itself is deleted, the space occupied by the file is released, and its contents are lost.

Write access (to the directory) is required to delete the directory entry.

If the file is open at the time the system call is made and the specified directory entry is the last link to the file, the directory entry is deleted immediately. The file itself is not deleted until the active process closes the file.

In order for a directory to be deleted, it must not

1. contain any files;
2. be the current directory of any user; or
3. be the root directory of a device.

system call: _divd
number: 54h
purpose: This call divides two unsigned 32-bit integers.

user access: all users

summary:

```
move.l  <dividend>, D1
move.l  <divisor>, D2
jsys    #_divd
move.l  D3,<quotient>
move.l  D4,<remainder>
```

calling parameters:

D1.L 32-bit unsigned dividend

D2.L 32-bit unsigned divisor

return parameters:

D3.L The D3.L register contains the 32-bit unsigned quotient.

D4.L The D4.L register contains the 32-bit unsigned remainder.

possible errors: ?ovflo

The _divd call returns D3.L = D1.L / D2.L, D4.L = D1.L % D2.L treated as unsigned 32-bit integers.

system call: _error
number: 1Ch
purpose: This call displays an error message.

user access: all users

summary:

```
move    <error number>, D0
move    <channel>, D1
lea     <pathname>, A0              (if needed)
lea     <alternate pathname>, A1    (if needed)
jsys    #_error
```

calling parameters:

D0 The D0 register contains the error number generated by a system call

D1 The D1 register contains the channel number. This channel receives the message and is usually set to stderr.

A0 Points to the pathname that will be displayed as the part of error message.

A1 Points to the alternate pathname that will be displayed as part of error message. The error number returned by a system call has bit 7 set if the error message should use the alternate pathname.

return parameters: none

possible errors: ?notopen

The _error call sends an error message to the specified channel. It should be called immediately after a system call that generated an error (or registers D0, A0, and A1 must be saved after the system call and restored prior to the _error call.

Errors may occur during calls to _error; this sets the carry flag.

system call:	_exchg
number:	0Ch
purpose:	This call exchanges the filenames of two open files.
user access:	all users

summary:

```
move    <channel number>, D1
move    <channel number>, D2
jsys    #_exchg
```

calling parameters:

D1 The D1 register contains the channel number of one file.

D2 The D2 register contains the channel number of the second file.

return parameters: none

possible errors: ?notopen

The _exchg call exchanges the filenames of two open files. After _exchg is executed, the two filenames remain associated with their original inodes, but the block pointers of the inodes are changed.

system call: _exec
number: 4Ch
purpose: This call executes a program.

user access: all users

summary:

```
lea     <argument list>, A1
lea     <pathname>, A0
jsys    #_exec
```

calling parameters:

A1 The A1 register points to a list of pointers. The list of pointers is terminated by a null pointer. Each pointer points to a null-terminated character string. Each string is an argument passed to the new program.

A0 The A0 register points to the pathname of the file to be executed. A null character terminates the pathname.

return parameters: none (does not return)

possible errors:

?notexist
?filaccess
?nomemory

The _exec call attempts to load the new program in a free memory area. If there is no memory available, the _nomemory error is returned.

All channels opened before the execution of the _exec call are passed to the new process.

system call: _exit
number: 46h
purpose: This call exits from a process.

user access: all users

summary:

```
move    <termination status>, D3
jsys    #_exit
```

calling parameters: D3 The D3 register contains the termination status to be passed back to the calling program.

0	termination OK
nonzero	abnormal termination

return parameters: none (does not return)

possible errors: none

The _exit call provides an exit from an active process. It closes all channels and unlocks all locks that the current process initiated.

The termination status is a user-defined value that the user wishes Cromix to pass back to the calling program. Normally, 0 (zero) indicates no error; any other value indicates an error. (The shell if -err construction tests the termination status of the last program executed.)

system call: _faccess
number: 26h
purpose: This call tests file access.

user access: all users

summary:

```
move    <access bits>, D2
lea     <pathname>, A0
jsys    #_faccess
```

calling parameters:

D2 The D2 register contains the access bits to be tested. These bits can be ORed together to test for various combinations of access privileges. These bits may be represented by:

AC_READ	read
AC_EXEC	execute
AC_WRIT	write
AC_APND	append

A0 The A0 register points to the pathname of the file to be tested. The pathname must be terminated by a null character.

return parameters:

The carry flag is reset (=0) if the file may be accessed as specified.

The carry flag is set and the D0.L register contains the error code _fileaccess if the file cannot be accessed as specified.

possible errors:

?badname
?fileaccess
?notexist

The _faccess call tests the access privileges of a file.

system call: _fchstat
number: 22h
purpose: This call changes the status of a file.

user access: see table

summary:

```
lea     <pathname>, A0
move    <status type>, D2
move    <new value>, D3
move    <access mask>, D4    (only for access)
lea     <buffer>, A1         (only for times)
jsys    #_cchstat
```

calling parameters:

A0 The A0 register points to the pathname of the file whose status is to be changed.

D2 The D2 register contains the status type to be changed.

For access privilege changes:

D3 The D3 register contains the new value of the specified status type.

D4 The D4 register contains the mask of the status bits to be changed:

AC_READ	read
AC_EXEC	execute
AC_WRIT	write
AC_APND	append

For time-changing calls:

A1 The register A1 points to a 6-byte buffer which contains the new time (year, month, day, hour, minutes, seconds).

For other status changes:

D3 The D3 register contains the new value.

return parameters: none

possible errors:

?filaccess
?priv
?notexist
?badname

The _fchstat call changes the access privileges associated with a file, the times associated with a file, the owner id of a file, or the group id of a file.

Table of Fchstat Calls

* Who	D2 Register	Status Type	Location of New Information
p	ST_OWNER	owner id	D3 = new value
p	ST_GROUP	group id	D3 = new value
p&o	ST_AOWNER	access owner	D3 = value, D4 = mask
p&o	ST_AGROUP	access group	D3 = value, D4 = mask
p&o	ST_AOTHER	access public	D3 = value, D4 = mask
p	ST_TCREATE	time created	A1 -> 6-byte buffer
p	ST_TMODIFY	time last modified	A1 -> 6-byte buffer
p	ST_TACCESS	time last accessed	A1 -> 6-byte buffer
p	ST_TDUMPED	time last dumped	A1 -> 6-byte buffer

*
p = privileged user
o = owner

system call: _fexec
number: 4Bh
purpose: This call forks and executes a program.

user access: all users

summary:

```
lea      <argument list>, A1
lea      <pathname>, A0
move     <signal mask>, D1
move     <signal values>, D2
jsys     #_fexec
move.l   D3, <new PID>
```

calling parameters:

A1 The A1 register points to a list of pointers. The list of pointers is terminated by a null pointer. Each pointer points to a null-terminated character string. Each string is an argument passed to the new program.

A0 The A0 register points to the pathname of the file to be executed. A null character terminates the pathname.

D1 The D1 register contains an 8-bit mask which indicates what signals to pass to the child (new) process. If a bit is reset (=0) then the corresponding bit in the D2 register is ignored. The child process will either ignore or be aborted by the signal corresponding to that bit, depending on whether the parent ignores or is aborted by the signal; if the parent process has provided a trapping routine (i.e., with the _signal call) the child process will again be aborted as it cannot inherit trapping routines. If a bit is set (=1), the corresponding bit of the D2 register determines what the child process does with the corresponding signal.

D2 If the corresponding bit in the D1 register is set (=1), the bit in the D2 register indicates the action to be taken by the child process when the corresponding signal is received. A bit that is reset (=0) causes the child process to abort when that signal is received. A bit that is set (=1) causes that signal to be ignored. The kill signal cannot be masked.

return parameters:

D3.L The D3.L register contains the child process id (PID) number.

possible errors:	?notexist ?filaccess ?nomemory ?badname ?usrtable

The _fexec call begins execution of a program and returns control to the calling program. This call is similar to the _exec call, except that a new process is created.

The child process inherits only the channels 0, 1, and 2 (if they are open), but not all open channels.

Notes

Only signals one through eight can be passed or masked in this call. Bit zero corresponds to signal one, bit one to signal two, and so on.

system call:	_flink	
number:	24h	
purpose:	This call establishes a link to a file.	
user access:	all users	
summary:	lea	<old pathname>, A0
	lea	<new pathname>, A1
	jsys	#_flink
calling parameters:	A0	The A0 register points to the existing file pathname. The pathname is terminated by a null character.
	A1	The A1 register points to the file pathname to be established (the new pathname). The pathname must be terminated by a null character.
return parameters:	none	
possible errors:	?badname ?isdir ?numlinks ?diraccess ?exists ?notexist	

The _flink call establishes a link to a file.

system call:	_fshell
number:	48h
purpose:	This call forks a shell process.
user access:	all users

summary:

```
lea      <argument list>, A1
move     <signal mask>, D1
move     <signal values>, D2
jsys     #_fshell
move.l   D3, <new PID>
```

calling parameters:

A1 The A1 register points to a list of pointers. The list of pointers is terminated by a null pointer. Each pointer points to a null-terminated character string. Each string is an argument passed to the new program.

D1 The D1 register contains an 8-bit mask which indicates what signals to pass to the child (new) process. If a bit is reset (=0) then the corresponding bit in the D2 register is ignored. The child process will either ignore or be aborted by the signal corresponding to that bit, depending on whether the parent ignores or is aborted by the signal; if the parent process has provided a trapping routine (i.e., with the _signal call) the child process will again be aborted, as it cannot inherit trapping routines. If a bit is set (=1), the corresponding bit of the D2 register determines what the child process does with the corresponding signal.

D2 If the corresponding bit in the D1 register is set (=1), the bit in the D2 register indicates the action to be taken by the child process when the corresponding signal is received. A bit that is reset (=0) causes the child process to abort when that signal is received. A bit that is set (=1) causes that signal to be ignored. The kill signal cannot be masked.

return parameters:

D3.L The D3.L register contains the child process id (PID) number.

possible errors: ?nomemory

The _fshell call initiates execution of a child shell process which acquires a new PID.

Options

These options are needed only when a program is calling a shell. They are not useful when a shell is called from the terminal.

The -c option indicates that the command line as a whole is passed to the shell. Shell will treat it as if it were typed from the terminal.

The -p option indicates that the command line being passed to the shell is already broken into separate arguments.

The -q option requests that the lines from a command file not be echoed to the terminal (standard output).

The -z option can be used when forking an interactive Shell (Shell with no arguments). This option causes the new Shell to ignore CONTROL-Z (End Of File). If the option is not set, a CONTROL-Z character will terminate the Shell.

Notes

Only signals one through eight can be passed or masked in this call. Bit zero corresponds to signal one, bit one to signal two, and so on.

The _fshell call expects its arguments to be in one of the following three forms:

Form 1 (passing command filenames)

```
A1  ->  arg 0  ->  "shell\0"
        arg 1  ->  arg 1 (a command filename)
        arg 2  ->  arg 2 (first argument for command)
        .
        .
        .
        0
```

Form 2 (passing a parsed argument list)

```
A1  ->  arg 0  ->  "shell\0"
        arg 1  ->  "-p\0"
        arg 2  ->  command name
        arg 3  ->  command's first argument
        arg 4  ->  command's second argument
        .
        .
        .
        0
```

Form 3 (passing a command line)

```
A1  ->  arg 0  ->  "shell\0"
        arg 1  ->  "-c\0"
        arg 2  ->  full command line
        0
```

system call:	_fstat
number:	20h
purpose:	This call returns the status of a file.
user access:	all users

summary:

```
lea     <pathname>, A0
move    <status type>, D2
lea     <buffer>, A1        (if necessary)
jsys    #_fstat
```

calling parameters:

A0 The A0 register points to the pathname of the file whose status is to be checked.

D2 The D2 register contains the request to the system for the desired information.

A1 The register A1 may point to a 6-byte or 128-byte buffer. Refer to the table.

return parameters: See table

possible errors: ?badname

The _fstat call returns file status information. Please refer to the following table of _fstat calls.

Table of Fstat Calls

D2 Register	Information Returned	Location of the Returned Information
ST_ALL	all of inode	A1 -> 128 byte buffer
ST_OWNER	owner idc	D3.L
ST_GROUP	group idc	D3.L
ST_AOWNER	access owner	D3.L
ST_AGROUP	access group	D3.L
ST_AOTHER	access public	D3.L
ST_FTYPE	file type	D3.L = IS_ORDIN IS_DIRECT IS_CHAR IS_BLOCK IS_PIPE
ST_SIZE	file size	D3.L
ST_NLINKS	number of links	D3.L
ST_INUM	inode number	D3.L
ST_TCREATE	time created	A1 -> 6 byte buffer
ST_TMODIFY	time last modified	A1 -> 6 byte buffer
ST_TACCESS	time last accessed	A1 -> 6 byte buffer
ST_TDUMPED	time last dumped	A1 -> 6 byte buffer

ST_DEVNO	device number	D3.L = major device #
		D4.L = minor device #

ST_DEVICE	device number	D3.L = major device #
		D4.L = minor device #

ST_PDEVNO	device number	D3.L = major device #
		D4.L = minor device #

ST_DEVNO returns the device numbers of the device specified in a device file. If the specified file is not a device file, ST_DEVNO returns zeroes. ST_DEVICE returns the device numbers of the device on which the specified file resides.

ST_PDEVNO returns the same value as ST_DEVNO except: for block device number zero the device number of the root device is returned; for character device number zero the device number of the user's terminal is returned.

system call: _getdate
number: 30h
purpose: This call returns the date.

user access: all users

summary:

```
jsys      #_getdate
move.l    D0, <weekday>
move.l    D1, <year>
move.l    D2, <month>
move.l    D3, <day>
```

calling parameters: none

return parameters:

D0.L The D0.L register contains the day of the week (1 represents Sunday, 2 represents Monday, etc.).

D1.L The D1.L register contains the year minus 1900. This means 1983 is represented as 83 and 2004 is 104.

D2.L The D2.L register contains the month (1 represents January, 2 represents February, etc.).

D3.L The D3.L register contains the day of the month (between 1 and 31).

possible errors: none

The _getdate call returns the current day as recorded by the Cromix system clock.

system call: _getdir
number: 02h
purpose: This call returns the current directory pathname.

user access: all users

summary:

```
lea     <buffer>, A0
jsys    #_getdir
```

calling parameters: A0 — The A0 register points to a 128 byte buffer for the pathname of the current directory.

return parameters: none

possible errors: none

The _getdir call returns the pathname of the current directory.

system call: _getgroup
number: 36h
purpose: This call returns the group id.

user access: all users

summary:

```
move     <id type>, D2
jsys     #_getgroup
move.l   D3, <group number requested>
```

calling parameters: D2 The D2 register contains a value indicating the type of identification desired.

ID_EFFECTIVE
ID_LOGIN
ID_PROGRAM

return parameters: D3.L The D3.L register contains the type of group identification requested.

possible errors: none

The _getgroup call returns the group id.

system call: _getmode
number: 12h
purpose: This call returns the characteristics of a character device.

user access: all users

summary:

```
move    <channel>, D1
move    <mode type>, D2
jsys    #_getmode
move.l  D3, <mode value>
```

calling parameters:

D1 The D1 register contains the channel number of the device.

D2 The D2 register contains the MODE TYPE to be tested.

return parameters:

D3.L The D3 register contains the value of the mode type specified by the D2 register.

possible errors: none

The _getmode call returns the characteristics of a character device. For more information, refer to the description of the **modeequ.asm** and **bmodeequ.asm** files in appendix A and the Mode utility in the Cromix-Plus User's Reference Manual.

system call: _getpos
number: 10h
purpose: This call returns a file pointer.

user access: all users

summary:

```
move      <channel number>, D1
jsys      #_getpos
move.l    D3, <file position>
```

calling parameters:

D1 The D1 register contains the channel number of the open file.

return parameters:

D3.L The D3.L register contains the current value of the file pointer. This is a 32-bit unsigned integer.

possible errors: ?notopen

The _getpos call returns the logical position of the file.

system call: _getprior
number: 38h
purpose: This call returns the priority of the calling process.

user access: all users

summary:

```
jsys     #_getprior
move.l   D3, <process priority>
```

calling parameters: none

return parameters: D3.L The D3.L register contains the priority number of the current process (-40 to +40).

possible errors: none

The _getprior call returns the priority number of the calling process. This number is within the range -40 (highest priority) to +40 (lowest priority).

system call:	_getproc	
number:	3Ah	
purpose:	This call returns the PID of the calling process.	
user access:	all users	
summary:	jsys move.l	#_getproc D3, <PID>
calling parameters:	none	
return parameters:	D3.L	The D3.L register contains the process id.
possible errors:	none	

The _getproc call returns the process id of the calling process.

system call: _gettime
number: 32h
purpose: This call returns the time.

user access: all users

summary:

```
jsys     #_gettime
move.l   D1, <hour>
move.l   D2, <minute>
move.l   D3, <second>
```

calling parameters: none

return parameters:

D1.L The D1.L register contains the hours portion of the current time based on a 24-hour clock.

D2.L The D2.L register contains the minutes portion of the current time. This is the number of minutes since the current hour started.

D3.L The D3.L register contains the seconds portion of the current time. This is the number of seconds since the current minute started.

possible errors: none

The _gettime call returns the current time as recorded by the Cromix system clock.

system call: _getuser
number: 34h
purpose: This call returns the user id of the current process.

user access: all users

summary:

```
move    <id type>, D2
jsys    #_getuser
move.l  D3, <user>
```

calling parameters: D2 The D2 register contains a value indicating the type of identification desired.

```
ID_EFFECTIVE
ID_LOGIN
ID_PROGRAM
```

return parameters: D3.L The D3.L register contains the type of id identification requested.

possible errors: none

The _getuser call returns the user id.

system call: _indirect

number: 51h

purpose: This call executes the system call identified by the number in the D0 register.

user access: all users

summary:

```
move    <call number>, D0
;all other registers as required by the call
jsys    #_indirect
```

calling parameters: D0 The D0 register contains the system call number.

return parameters: According to system call.

possible errors: According to system call.

The _indirect call executes a system call identified by the value in the D0 register. Note that this use of the D0 register prevents the _error and _wrbyte system calls from being used with the _indirect system call.

system call:	_kill
number:	41h
purpose:	This call sends a signal to a process.
user access:	all users processes initiated by the user privileged user any process

summary:

```
move    <signal type>, D2
move    <process id>, D3
jsys    #_kill
```

calling parameters:

D2	The D2 register contains the type of signal to be sent.
D3	The D3 register contains the process id of the process to which the signal is sent.

return parameters: none

possible errors:

?priv
?noproc
?badcall

The _kill call sends a signal to a process. When any signal is received by a process, the process is aborted unless the _signal system call specifies that a subroutine be executed or the signal be ignored.

When a signal is received, unless it is ignored, an unsatisfied request for input or output from a character device is canceled. Examples are reading a buffered line from a console or writing a line to the printer.

If a signal is sent to process 0, the same type of signal is sent to all processes that belong to the user invoking the call.

If the user is a privileged user and a SIGUSER signal is sent to process 1, system shutdown is initiated.

If a SIGABORT signal is sent to process 1, the /etc/ttys file is reexamined. If an entry has a 0 in the leftmost column, the appropriate terminal is logged off and all of its processes are terminated. If an entry shows a 1 in that column, the terminal is logged in if it is not already logged in.

system call: _lock
number: 3Eh
purpose: This call assists in implementing interprocess communications.

user access: all users

summary:

```
move    <lock type>, D2
move    <lock length>, D3
lea     <lock sequence>, A0
jsys    #_lock
```

calling parameters:

D2 The D2 register contains the type of lock to be implemented.

bit 0 If bit 0 of the D2 register contains 0, the lock may not be shared; a 1 indicates the lock may be shared. A shared lock may be used by more than one process.

bit 1 If bit 1 of the D2 register contains 0, then the lock is unconditional; a 1 indicates that the lock is conditional. If a conditional lock fails, a _locked error is returned. If an unconditional lock fails, the process is put to sleep until the lock does not fail. Failure implies that the lock sequence matches the lock sequence of a prior lock still in effect in one of the following ways:

1. A nonsharable lock was requested when a matching lock already existed.

2. A sharable lock was requested when a nonsharable matching lock already existed.

3. The lock table is full. This returns a _lcktable error to the process. There is space for 16 locks.

bit 2 If bit 2 of the D2 register contains 0, the lock sequence is completely determined by the user. If bit 2 is set, the lock sequence is guaranteed to be unique. Only processes forked by the _fork system call will be able to produce the same locking sequence.

	D3	The D3 register contains the length of the locking sequence. This must be a number between 1 and 16.
	A0	The A0 register points to the locking sequence of 16 or fewer bytes.
return parameters:	none	
possible errors:	?locked ?deadlock ?lcktable	

system call: _makdev
number: 00h
purpose: This call creates a new name for a device.

user access: privileged user

summary:

```
move    <type of device>, D2
move    <major device #>, D3
move    <minor device #>, D4
lea     <pathname>, A0
jsys    #_makdev
```

calling parameters:

D2 The D2 register indicates the type of device:

IS_BLOCK	block device
IS_CHAR	character device

D3 The D3 register contains the major device number.

D4 The D4 register contains the minor device number.

A0 The A0 register points to the new pathname for the device. The pathname must be terminated by a null character.

return parameters: none

possible errors:

?badname
?exists

The _makdev call assigns a label to an existing device in the operating system.

system call:	_makdir	
number:	01h	
purpose:	This call creates a new directory.	
user access:	all users	
summary:	lea	<pathname>, A0
	jsys	#_makdir
calling parameters:	A0	The A0 register points to the pathname of the new directory. The pathname must be terminated by a null character.
return parameters:	none	
possible errors:	?badname ?exists	

The _makdir call creates a new directory.

system call: _memory
number: 50h
purpose: This call allocates or deallocates memory.

user access: all users

summary:

```
move.l   <mask>, D1               (if allocating)
move     <type>, D2
move.l   <size>, D3
lea      <memory pointer>, A0     (if deallocating)
jsys     #_memory
move.l   A0, <memory pointer>     (if allocating)
```

calling parameters:

D1.L The D1.L register contains a value which is used only for allocation. The normal value is zero. A nonzero value restricts the address of the memory being allocated. The pointer returned, if masked with the given mask, will be zero. For example, to get memory at a 64K boundary, specify the mask as 0xffff.

D2 The D2 register contains a value indicating the type of action desired.

0	allocate memory
1	deallocate memory

D3.L Size of memory (in bytes).

A0 Pointer to existing memory (if it is to be deallocated).

return parameters:

A0 The A0 register contains the pointer to the memory obtained (if allocating).

possible errors:

?nomemory
?badvalue

For type = 0 the amount of memory defined by the D3.L register will be obtained from the system and the pointer to it returned in the A0 register. For type = 1 the number of bytes defined by D3.L register and pointed to by A0 register will be deallocated (returned to the system pool). Only the memory obtained by the _memory system call should be deallocated. Memory is allocated and deallocated in 4K chunks. Two consecutive calls to request memory do not guarantee that the pieces obtained will be consecutive or in any particular position relative to the position of the user code.

system call: _mount
number: 04h
purpose: This call enables access to a file system.

user access: privileged user

summary:

```
move    <type of access>, D2
lea     <dummy pathname>, A0
lea     <block device pathname>, A1
jsys    #_mount
```

calling parameters:

D2 The D2 register indicates the desired access:

0 read/write
1 read only

A0 The A0 register points to a buffer containing the pathname of the dummy file in which the file system is to be mounted. The pathname must be terminated by a null character.

A1 The A1 register points to a buffer containing the pathname of the block device which contains the file system to be mounted. The pathname must be terminated by a null character.

return parameters: none

possible errors:

?mttable
?fsbusy
?notblk
?badname
?notexist

The _mount call declares that a file system is to be mounted on a specified device. References to the file system pathname refer to the root file of the mounted file system.

The dummy file pathname is the file system pathname while the file system remains mounted. When the system is unmounted, the name reverts to dummy status.

system call: _mult
number: 53h
purpose: This call multiplies two unsigned 32-bit integers.

user access: all users

summary:

```
move.l  <multiplicand>, D1
move.l  <multiplicator>, D2
jsys    #_mult
move.l  D3,<product>
```

calling parameters:

D1.L	32-bit unsigned multiplicand
D2.L	32-bit unsigned multiplicator

return parameters:

D3.L	The D3 register contains the 32-bit unsigned product.

possible errors: ?ovflo

The _mult call returns D3.L = D1.L * D2.L treated as unsigned 32-bit integers.

system call: _open
number: 09h
purpose: This call opens a file for access.

user access: all users

summary:

```
lea      <pathname>, A0
move     <access mode>, D2
move     <exclusive mask>, D3
jsys     #_open
move.l   D1, <channel>
```

calling parameters:

A0 The A0 register points to a buffer containing the pathname of the file to be opened. The pathname must be terminated by a null character.

D2 The D2 register contains the access mode value for opening the file. The following labels represent the values of the D2 register required to establish the desired access mode. The specified access mode is applicable to the current process.

Nonexclusive access values:

OP_READ	read only
OP_WRITE	write only
OP_RDWR	read/write
OP_APPEND	append

Exclusive access values:

OP_XREAD	read only
OP_XWRITE	write only
OP_XRDWR	read/write
OP_XAPPEND	append

If exclusive access is desired, one of the four exclusive access values listed above must be loaded into the D2 register. This, in conjunction with the desired exclusion bit(s) in the D3 register, denies other users access.

D3 The D3 register contains the mask for exclusive access. It is inspected only if the D2 register indicates exclusive access. Each of the specified bits must be set to prevent the file from being opened by another process for the specified access. (For example, ^OP_READ indicates that no other process may open the file with the read access. This does not exclude another process from opening the file for read/write access. To exclude all reads,

^OP_READ and ^OP_RDWR must be ored together.) The following values may be ored together to set more than one bit.

Exclusive access bits:

^OP_READ	exclude read
^OP_WRITE	exclude write
^OP_RDWR	exclude read/write
^OP_APPEND	exclude append

return parameters: D1.L The D1.L register contains the channel number that the system assigned to the file.

possible errors:

?filtable
?badname
?diraccess
?isdir

The _open call assigns a channel number to the specified file. The user is then allowed to read from and/or write to the file.

system call:	_pause
number:	44h
purpose:	This call suspends process execution and waits for a signal.
user access:	all users
summary:	jsys #_pause
calling parameters:	none
return parameters:	none
possible errors:	none

The _pause call suspends execution of the current process until a signal generated by the _kill or _alarm system call is received.

system call:	_pipe	
number:	0Eh	
purpose:	This call creates a pipe.	
user access:	all users	
summary:	jsys	#_pipe
	move.l	D1, <reading side>
	move.l	D2, <writing side>
calling parameters:	none	
return parameters:	D1.L	The D1.L register contains the number of the channel into which the data is read out from the pipe.
	D2.L	The D2.L register contains the number of the channel from which the data is written into the pipe.
possible errors:	?toomany	

The _pipe system call returns two channel numbers. One channel number is the writing end of the pipe, the other channel is the reading end of the pipe. You will end up with two processes, one holding the writing end of the pipe, the other one holding the reading end of the pipe. The writing process can then write into the pipe without much ado. If it starts to overfill the pipe, Cromix will put the writing process to sleep until the reading process makes room in the pipe. By reading from the pipe, the reading process will wake up the writing process if it fell asleep. If the reading process reads so far that the pipe becomes empty, the reading process will go to sleep until the writing process starts writing and wakes it up. The only time a problem can arise is if the reading process dies. If the writing process tries to write to the other end of the pipe while the reading process is dead, Cromix will kill the writing process by sending the sigpipe signal.

The problem that remains to be solved is how to pass one end of the pipe to another process. There are two facts which are used to achieve this end:

1. Whenever the system needs a new channel number, it will pick up the lowest available number.

2. Whenever a process is forked, the child process will inherit the channels 0, 1, and 2 (stdin, stdout, and stderr) from the parent process.

Suppose a process wants to fork a child process and talk to it through a pipe. More specifically, the parent process will do the writing, the process will do the reading. If some other setup is desired, the strategy described below can easily be changed or extended. The solution is to ensure that the child's stdin channel is going to be the reading end of the pipe.

1. Create a pipe using the _pipe system call. Let the reading and the writing channels be called rchan and wchan, respectively.

2. Make a duplicate stdin channel (call it oldstdin), so as not to loose it completely.

3. Close the stdin channel.

4. Duplicate the rchan channel. Step 3 above guarantees that the lowest channel number available is stdin, so you do not need to pay attention to what the duplicate channel is. You know it is going to be stdin again.

5. Close the rchan channel. You do not need it anymore, as the reading end of the pipe is now the stdin channel.

6. Fork the child process (_fexec or _fshell). The child process will inherit the first three channels from the parent process, which means that the child's stdin channel is the reading end of the pipe.

7. Here the cleanup operation starts. The parent process wants its own stdin channel back, so close the stdin channel to ensure that stdin is the lowest free channel.

8. Duplicate the oldstdin channel. Step 7 guarantees you are going to get stdin again, so you do not need to pay attention to what you get.

The parent process now has its own channels back. Whatever it writes to the wchan channel will be received by the child process on its stdin channel. If the parent process wants to signal the end of the file to the child process it can simply close the wchan channel. When the child process reaches the end of pipe and finds the writing end of the pipe closed, the child process will get an end-of-file on the read operation.

After the parent process has written to the pipe everything it intended, the parent process should wait until the child terminates. Eventually, the child process will hit the end of the pipe and presumably terminate. The parent process can then close the wchan channel, and the pipe completely disappears.

If the parent process wants to read what the child process is going to write, the above scheme can be easily modified:

1. Create a pipe using the _pipe system call. Let the reading and the writing channels be called rchan and wchan, respectively.

2. Make a duplicate stdout channel (call it oldstdout) so as not to loose it completely.

3. Close the stdout channel.

4. Duplicate the wchan channel. Step 3 above guarantees that the lowest channel number available is stdout, so you do not need to pay attention to what the duplicate channel is. You know it is going to be stdout again.

5. Close the wchan channel. You do not need it anymore, as the writing end of the pipe is now the stdout channel.

6. Fork the child process (_fexec or _fshell). The child process will inherit the first three channels from the parent process, which means that the child's stdout channel is the writing end of the pipe.

7. Here the cleanup operation starts. The parent process wants its own stdout channel back, so close the stdout channel to ensure that stdout is the lowest free channel.

8. Duplicate the oldstdout channel. Step 7 guarantees you are going to get stdout again, so you do not even pay attention to what you get.

The parent process has now its own channels back. Whatever the child process writes to it's stdout channel will be available on the rchan channel to the parent process. If the child process wants to signal the end of file to the parent process it can simply close the stdout channel (normally by jsys _exit). When the parent process reaches the end of pipe and finds the writing end of the pipe closed, the parent process will get an end-of-file on the read operation.

There are many other variations possible, including two-way communication. For this, you need two independent pipes: the final result is achieved by merging the above two schemes.

system call:	_printf
number:	1Bh
purpose:	This call generates formatted output.
user access:	all users

summary:

```
move    <channel>, D1
lea     <control string>, A0
;push all arguments, last first
jsys    #_printf
;pop all arguments
```

calling parameters:

D1	The D1 register contains the output channel number.
A0	The A0 register points to the null-terminated control string.
stack	All arguments must be pushed onto the stack before the call (last argument pushed first) and popped off the stack after the call.

return parameters: none

possible errors: ?notopen

The _printf call generates formatted output.

The null-terminated control string is composed of regular characters and conversion specifications. Regular characters are copied directly to the output file. Conversion specifications are introduced by a percent (%) sign and terminated by the conversion character itself.

The conversion specifications have the following format:

```
%-xxx.yyyL,z
```

The percent sign and the conversion character itself (z) are required; all conversion-specification characters in between are optional.

A minus sign may follow the percent sign. If it is included, the argument is left justified. Otherwise the argument is right justified.

Following this may be two strings of digits separated by a period (represented by xxx.yyy). The first of these numbers represents the minimum field width.

If it is not included, the minimum field width is assumed to be zero. The second of these numbers represents the maximum field width. If it is not included, the maximum field width is as large as necessary.

If the character L (or l) appears after this, it signifies that the argument is a long (32-bit) number. If it is absent, the argument is assumed to be short (16 bits).

If a comma appears before the decimal conversion character, commas appear in the output (as in 1,000,000).

The conversion character itself (represented by z) may be any of the following:

d The argument is converted to a decimal number.

u The argument is converted to an unsigned decimal number.

x The argument is converted to an unsigned hexadecimal number.

c The argument is assumed to be a single character. When this argument is pushed onto the stack, the character must be in the low-order byte of the word pushed.

s The argument is assumed to be a character string. A (4 byte) pointer to this string must be pushed onto the stack instead of the string itself.

system call: _ptrace
number: 4EH
purpose: This call runs a process debugger. Actual function depends on the function value (refer to the ptrace.h header file)

user access: all users

summary:

```
move    <function code>, D1
move    <pid>, D2
lea     <address>, A0
lea     <data>, A1
move.l  <count>, D3
jsys    #_ptrace
```

calling parameters:

D1.W The D1.W register contains the function code of the _ptrace call.

D2.W The D2.W register contains the process id of the process being debugged (child pid).

A0.L The A0.L register contains the address in the current (parent) process where information is read from or written to.

A1.L The A1.L register contains the address in the child process (absolute address) where information is read from or written to.

D3.L The D3.L register contains the number of bytes to be transferred.

return parameters: none

possible errors: ?badvalue
?hoproc

The _ptrace system call has the following subfunctions (selected by the value in the D1.W register:

P_START The parent process (debugger) issues this call to notify the system that the next fexec (fshell, fork) system call will fork a debugged process. The debugged process does not start execution by itself; it waits for the parent process to issue a P_RUN, P_TRACE, or P_TERM ptrace function. (The debugged process behaves as if it just hit a breakpoint). The pid, address, data, and count arguments are not used.

P_RDSEQ When the debugged process is in the suspended state, this call reads D3.L bytes from the (absolute) A1.L address of the D2.W process into the (absolute) A0.L parent address. The specified process must be started with the P_START function before the fexec call.

P_WRSEQ When the debugged process is in the suspended state, this call writes D3.L bytes to the (absolute) A1.L address of the D2.W process from the (absolute) A0.L parent address. The specified process must be started with the P_START function before the fexec call.

P_RDSTA When the debugged process is in the suspended state, this call reads all of the D2.W process registers (see ptrace.h) into the (absolute) A0.L parent address. The A1 and D3 registers are not used with this call. The specified process must be started with the P_START function before the fexec call.

P_WRSTA When the debugged process is in the suspended state, this call writes all of the D2.W process registers (see ptrace.h) from the (absolute) A0.L parent address. A1 and D3 registers are not used with this call. The specified process must be started with the P_START function before the fexec call.

P_RUN When the debugged process is in the suspended state, this call restarts the D2.W process. The parent process normally installs breakpoints before issuing this call. Breakpoints can be installed by patching the child code with the TRAP #5 instruction. When the child process execute the TRAP #5 instruction, it goes into the suspended state, and the system notifies the parent process with a sigtrace signal. The specified process must be started with the P_START function before the fexec call.

P_TRACE When the debugged process is in the suspended state, this call restarts the D2.W process for the duration of one instruction. After one instruction is executed, the system notifies the parent process with a sigtrace signal. The specified process must be started with the P_START function before the fexec call.

P_TERM When the debugged process is in the suspended state, this call terminates the D2.W process. The specified process must be started with the P_START function before the fexec call.

system call: _rdbyte
number: 16h
purpose: This call reads a byte.

user access: all users

summary:

```
move    <channel>, D1
jsys    #_rdbyte
move.l  D0, <value read>
```

calling parameters: D1 The D1 register contains the channel number of the file.

return parameters: D0.L The D0.L register contains the byte read.

possible errors:

?notopen
?filaccess
?ioerror
?endfile
?signal

The _rdbyte call reads the next sequential byte going toward the end of the file from the open file on the channel specified.

To eliminate the need for the input to be terminated by a RETURN character, set the device mode to "raw".

system call: _rdline
number: 18h
purpose: This call reads a line.

user access: all users

summary:

```
move    <channel>, D1
move.l  <maximum bytes>, D3
lea     <buffer>, A0
jsys    #_rdline
move.l  D3, <bytes read>
```

calling parameters:

D1 The D1 register contains the channel number of the file.

D3.L The D3 register contains the maximum number of bytes to be read with this call.

A0 The A0 register points to the buffer in which the line is returned.

return parameters:

D3.L The D3 register contains the number of bytes read, including the line terminator.

possible errors:

?notopen
?filaccess
?ioerror
?endfile
?signal

The _rdline call reads a line, or a number of sequential bytes moving towards the end of file, from the file opened on the specified channel.

The buffer is filled with bytes until an end-of-line indicator (a linefeed or null character) is encountered.

system call:	_rdseq
number:	14h
purpose:	This call reads the specified number of bytes.
user access:	all users

summary:

```
move     <channel>, D1
move.l   <byte count>, D3
lea      <buffer>, A0
jsys     #_rdseq
move.l   D3, <bytes read>
```

calling parameters:

D1	The D1 register contains the channel number of the file.
D3.L	The D3 register contains the number of sequential bytes to be read from the current position of the file pointer.
A0	The A0 register points to the buffer where the bytes are returned.

return parameters:

D3.L	The D3 register contains the actual number of bytes read.

possible errors:

?notopen
?filaccess
?ioerror
?endfile
?signal

The _rdseq call reads the next specified number of bytes, moving towards the end of file, from the file opened on the specified channel.

system call:	_setdate
number:	31h
purpose:	This call changes the date.
user access:	privileged user

summary:

```
move    <year>, D1
move    <month>, D2
move    <day of the month>, D3
jsys    #_setdate
```

calling parameters:	D1	The D1 register contains the year minus 1900. For example, 1983 is represented as 83 and 2004 is 104.
	D2	The D2 register contains the month (1 represents January, 2 represents February, etc.).
	D3	The D3 register contains the day of the month (between 1 and 31).
return parameters:	none	
possible errors:	?priv	

The _setdate call changes the Cromix system clock to the date specified. The parameters are binary numbers.

system call: _setdir
number: 03h
purpose: This call changes the current directory.

user access: all users

summary:

```
lea     <buffer>, A0
jsys    #_setdir
```

calling parameters: A0 The A0 register points to the new directory pathname. The pathname must be terminated by a null character.

return parameters: none

possible errors: ?notdir
?diraccess

The _setdir call changes the current directory to the one specified.

system call:	_setgroup
number:	37h
purpose:	This call changes the group id.
user access:	all users

summary:

```
move    <type of id to change>, D1
move    <new id type>, D2
move    <new id number>, D3
jsys    #_setgroup
```

calling parameters:

D1 The D1 register contains the type of id to be changed.

ID_EFFECTIVE
ID_LOGIN
ID_PROGRAM

D2 The D2 register indicates the value of the id type specified by the D1 register. This value may be the value of the other id types or the value specified by the D3 register.

ID_EFFECTIVE
ID_LOGIN
ID_PROGRAM
ID_D3

D3 If the D2 register contains ID_D3, the D3 register must contain a 16-bit id number.

return parameters: none

possible errors: ?priv

The _setgroup call changes the group id of the current process to the one specified. This call may be invoked only by a privileged user when the D2 register contains the value ID_D3.

system call: _setmode
number: 13h
purpose: This call changes the characteristics of a character device.

user access: all users

summary:

```
move    <channel>, D1
move    <mode type>, D2
move.l  <new value>, D3
move    <mask>, D4
jsys    #_setmode
move.l  D3, <old value>
```

calling parameters:

D1 The D1 register contains the channel number of the opened device.

D2 The D2 register contains the MODE TYPE to be set. The D2 register may be loaded with one of the mode types listed below.

D3.L The D3 register contains the new value of the mode type specified by the D2 register. Refer to the table below.

D4 The D4 register, in MD_MODE1, MD_MODE2, and MD_MODE3, is a mask indicating which characteristics to change.

return parameters:

D3.L The D3 register contains the previous value of the mode type specified by the D2 register.

possible errors: ?badvalue

The _setmode call changes the characteristics of a character device. For more information, refer to the **modeequ.asm** and **bmodeequ.asm** files in appendix A and to the description of the Mode utility in the Cromix-Plus User's Reference Manual.

system call: _setpos
number: 11h
purpose: This call changes the position of the file pointer.

user access: all users

summary:

```
move    <channel number>, D1
move    <mode>, D2
move.l  <file pointer>, D3
jsys    #_setpos
```

calling parameters:

D1 The D1 register contains the channel number of the open file.

D2 The D2 register contains the mode. This is the location from and direction to which the file pointer is established.

FWD_BEGIN	forward from the beginning of the file
FWD_CURRENT	forward from the current position
FWD_END	forward past the end of file
BAK_CURRENT	backward from the current position
BAK_END	backward from the end of file

D3.L The D3.L register contains the position change of the file pointer. This value is 32 bits. It should be nonnegative.

return parameters: none

possible errors:

?notopen
?notblk
?filaccess

The _setpos call changes the file pointer position to the specified logical byte position.

system call:	_setprior	
number:	39h	
purpose:	This call returns the priority of the calling process.	
user access:	all users	
summary:	move	<priority number>, D3
	jsys	#_setprior
calling parameters:	D3	The D3 register contains the new priority number (-40 to 40).
return parameters:	none	
possible errors:	?priv	

The _setprior call changes the current process priority as specified by the D3 register. The priority number must be between -40 (the highest priority) and 40. Only a privileged user may set a priority number between -40 and -1. The default priority assigned by the operating system is 0.

system call: _settime
number: 32h
purpose: This call changes the time.

user access: privileged user

summary:

```
move    <hours>, D1
move    <minutes>, D2
move    <seconds>, D3
jsys    #_settime
```

calling parameters:

D1 The D1 register contains the hours portion of the current time based on a 24-hour clock.

D2 The D2 register contains the minutes portion of the current time. This is the number of minutes since the current hour started.

D3 The D3 register contains the seconds portion of the current time. This is the number of the seconds since the current minute started. .

return parameters: none

possible errors: ?priv

The _settime call changes the Cromix system clock to the time specified. The parameters are binary numbers.

system call: _setuser
number: 35h
purpose: This call changes the user id.

user access: all users

summary:

```
move    <type of id to change>, D1
move    <new id type>, D2
move    <new id number>, D3
jsys    #_setuser
```

calling parameters:

D1 The D1 register contains the type of id to be changed.

ID_EFFECTIVE
ID_LOGIN
ID_PROGRAM

D2 The D2 register indicates the value of the id type specified by the D1 register. This value may be the value of the other id types or the value specified by the D3 register.

ID_EFFECTIVE
ID_LOGIN
ID_PROGRAM
ID_D3

D3 If the D2 register contains ID_D3, the D3 register must contain a 16-bit id number.

return parameters: none

possible errors: ?priv

The _setuser call changes the user id of the current process to the one specified. This call may be invoked only by a privileged user when the D2 register contains the value ID_D3.

system call:	_shell	
number:	49h	
purpose:	This call initiates a shell process.	
user access:	all users	
summary:	lea	<argument list>, A1
	jsys	#_fexec
calling parameters:	A1	The A1 register points to a list of pointers. The list of pointers is terminated by a null pointer. Each pointer points to a null-terminated character string. Each string is an argument passed to the new program.
return parameters:	none (does not return)	
possible errors:	?nomemory	

The _shell call initiates execution of a shell process. A new PID is not generated.

Options

These options are needed only when a program is calling a shell. They are not useful when a shell is called from the terminal.

The -c option indicates that the command line as a whole is passed to the shell. The shell will treat it as if it was typed from the terminal.

The -p option indicates that the command line being passed to the shell is already broken into separate arguments.

The -q option requests that the lines from a command file not be echoed to the terminal (standard output).

The -z option can be used when forking an interactive Shell (Shell with no arguments). This option causes the new Shell to ignore CONTROL-Z (End of file). If the option n is not set the CONTROL-Z character will terminate the Shell.

Notes

The _shell call expects its arguments to be in one of the following three forms:

Form 1 (passing command filenames)

```
A1  ->  arg 0  ->  "shell\0"
        arg 1  ->  arg 1 (a command filename)
        arg 2  ->  arg 2 (first argument for command)
        .
        .
        .
        0
```

Form 2 (passing a parsed argument list)

```
A1  ->  arg 0  ->  "shell\0"
        arg 1  ->  "-p0"
        arg 2  ->  command name
        arg 3  ->  command's first argument
        arg 4  ->  command's second argument
        .
        .
        .
        0
```

Form 3 (passing a command line)

```
A1  ->  arg 0  ->  "shell\0"
        arg 1  ->  "-c\0"
        arg 2  ->  full command line
        0
```

system call: _signal
number: 40h
purpose: This call sets up a process to receive a signal.

user access: all users

summary:

```
move    <type of signal>, D2
lea     <execution address>, A0
jsys    #_signal
move.l  A0, <old trap address>
```

calling parameters:

D2 The D2 register contains the type of signal.

SIGABORT	CNTRL-C signal
SIGUSER	user-specifiable key
SIGKILL	kill signal (not catchable)
SIGTERM	terminate signal
SIGALARM	alarm clock signal
SIGPIPE	broken pipe
SIGHANGUP	modem hangup signal

A0 The A0 register contains the program address to which control is transferred. If the A0 register contains 00000000, the process aborts upon receipt of the specified signal; if A0 contains 0000001, the signal is ignored.

return parameters:

A0 The A0 register contains the previous execution address.

possible errors:

?badcall
?signal

If the _signal call has been used to set up a subroutine address, control is passed to the subroutine at the address specified when the signal is received. The program returns to the point of execution where the signal was received on encountering an RTS instruction. Further signals of the same kind will be ignored unless _signal is used to set up the address again. Note that trap routines must preserve complete system status (all registers, including CCR).

system call: _sleep
number: 42h
purpose: This call puts a process to sleep.

user access: all users

summary:

```
move.l  <number of seconds to sleep>,D3
jsys    #_sleep
move.l  D3,<number of seconds left>
```

calling parameters: D3.L The D3.L register contains the number of seconds the process is to sleep.

return parameters: D3.L The D3.L register returns the number of seconds left if sleeping was aborted by a signal.

possible errors: none

The _sleep system call is used to put a process to sleep for a specified number of seconds. This frees processor time for other processes.

system call:	_trunc
number:	0Dh
purpose:	This call truncates an open file.
user access:	all users

summary:

```
move    <channel>, D1
jsys    #_trunc
```

calling parameters:	D1	The D1 register contains the channel number of the open file.
return parameters:	none	
possible errors:	?notopen	

The _trunc system call deletes the file from the current file pointer position through the end of file (or extends the file to the current position). This system call is mainly used to truncate a file to zero length.

system call: _uchstat
number: 29H
purpose: This call changes the status of a process.

user access: privileged user

summary:

```
move    <process id>, D1
move    <status type>, D2
move    <new value>, D3
jsys    #_uchstat
```

calling parameters:

D1 The D1 register contains the process id of the selected process. Zero means the current process.

D2 The D2 register contains the status type to be changed.

D3 The D3 register contains the new value of the specified status type.

return parameters: none

possible errors: ?noproc
?priv

The _uchstat call changes the process table information of the process identified by the process id. Process id zero refers to the current process. Only a privileged user can change the status of processes not his own.

Table of Uchstat Calls

D2 Register	**Status Type**	**Location of New Information**
USR_CTTY	controlling tty	D3.L = new value
USR_PRIOR	process priority	D3.L = new value
USR_TERM	termcap ident	D3.L = new value

system call: _unlock
number: 3Fh
purpose: This call is used to unlock a locking sequence.

user access: all users

summary:

```
move        <lock type>, D2
move        <lock length>, D3
lea         <lock sequence>, A0
jsys        #_unlock
```

calling parameters:

D2 The D2 register must contain the same value as it contained when the corresponding _lock system call was executed.

D3 The D3 register must contain the same value as it contained when the corresponding _lock system call was executed.

A0 The A0 register must contain the same value as it contained when the corresponding _lock system call was executed.

return parameters: none

possible errors:

The _unlock call unlocks a locking sequence that was locked by the _lock system call. Please refer to _lock system call for more information.

system call: _unmount
number: 05h
purpose: This call disables access to a file system.

user access: privileged user

summary:

```
move        <eject flag>, D2
lea         <block device pathname>, A0
jsys        #_unmount
```

calling parameters:

D2 If the D2 register contains a 1, the diskette that is unmounted is ejected. If D2 contains a 0, the diskette is not unmounted.

A0 The A0 register points to a buffer containing the pathname of the block device which contains the file system to be unmounted. The pathname must be terminated by a null character.

return parameters: none

possible errors:

?notmount
?fsbusy
?badname
?notexist

The _unmount call, used in conjunction with _mount, declares that the device no longer has the previously specified file system.

When the system is unmounted, the file system pathname reverts to being a dummy pathname.

system call:	_update	
number:	52h	
purpose:	This call updates all open files.	
user access:	all users	
summary:	jsys	#_update
calling parameters:	none	
return parameters:	none	
possible errors:	?ioerror	

The _update call causes all open files to be updated with the current contents of their buffers. This is done automatically upon closing a file.

system call: _ustat
number: 28H
purpose: This call returns the status of a selected process.

user access: all users

summary:

```
move    <process id>, D1
move    <status type>, D2
jsys    #_ustat
move.l  D3,<status value>
```

calling parameters

D1 The D1 register contains the process id of the selected process. Zero means the current process.

D2 The D2 register contains the request to the system for the desired information.

return parameters: See table

possible errors: ?noproc
?priv

The _ustat call returns process status information. The process id is used to identify the process (pid zero selects the current process). Only a privileged user can read the status of processes not his own.

Table of Ustat Calls

D2 Register	**Information Returned**	**Location of the Information Returned**
USR_CTTY	controlling tty	D3.L
USR_PRIOR	process priority	D3.L
USR_PARENT	parent process id	D3.L
USR_MEMP	program address	D3.L
USR_MEMS	total memory size	D3.L
USR_TIME	process time (ms)	D3.L
USR_CTIME	children time (ms)	D3.L
USR_USER	process owner	D3.L
USR_CTIME	process group	D3.L
USR_TERM	termcap ident	D3.L

system call:	_version
number:	55h
purpose:	This call returns the operating system version number.
user access:	all users

summary:

```
jsys      #_version
move.l    D3, <version number>
```

calling parameters:	none	
return parameters:	D3.L	The D3.L register contains the Cromix Operating System version number.
possible errors:	_corrupt	

The _version call returns the version number of the operating system.

Note

The version number is encoded in BCD. The version number 20.24, for example, is returned as 00002024h.

system call: _wait
number: 45h
purpose: This call waits for the termination of a child process.

user access: all users

summary:

```
move    <conditional flag>, D1
move    <process ID>, D3
jsys    #_wait
move.l  D3, <child PID>
move.l  D2, <termination status>
move.l  D1, <signal number >
```

calling parameters:

D1 If the D1 register equals zero, the call will not return until a child process has terminated.

If the D1 register equals one, this call returns immediately. An error is returned if no child process has terminated.

D3 If the D3 register contains a zero, this call waits for the termination of any child process.

If the D3 register is set equal to a process id (PID) number, this call waits for the termination of the specified process.

return parameters:

D3.L The D3.L register contains the child process id number.

D2.L The D2.L register contains the process termination status returned by the _exit system call.

D1.L The D1.L register contains the system termination status. If the D1 register equals zero, the child process was terminated through _exit. Otherwise, the D1 register contains the signal number of the signal that caused the termination and the D2 register is undefined.

possible errors: ?nochild

The _wait call informs the parent process when a child process is no longer active.

All processes created by forking (i.e., _fshell or _fexec) will remain in the process table after termination with a process status of 'T' until the _wait system call is made for the child's PID. The wait call must be made after the child has terminated.

If the call is made before the child process terminates, and the option to not wait until termination is selected, the child process will remain in the process table as terminated until the _wait call is made again. This means that if the 'no wait' option is selected, the _wait call should be made periodically until no error is returned.

system call: _wrbyte
number: 17h
purpose: This call writes a byte.

user access: all users

summary:

```
move    <channel>, D1
move.b  <byte>, D0
jsys    #_wrbyte
```

calling parameters:

D1 The D1 register contains the channel number of the file.

D0 The D0 register contains the byte to be written.

return parameters: none

possible errors:

?notopen
?filaccess
?ioerror

The _wrbyte call writes a byte to the file opened on the specified channel. The byte is written at the current file position. Note that this may overwrite information previously written to the file.

system call: _wrline
number: 19h
purpose: This call writes a line.

user access: all users

summary:

```
move    <channel>, D1
lea     <buffer>, A0
jsys    #_wrline
move.l  D3, <bytes written>
```

calling parameters:

D1 The D1 register contains the channel number of the file.

A0 The A0 register points to the buffer where the line to be written is stored.

return parameters:

D3.L The D3.L register contains the number of bytes actually written.

possible errors:

?notopen
?filaccess
?ioerror

The _wrline call writes a line to the file opened on the specified channel. The bytes are written at the current file position. Note that this may overwrite information previously written to the file.

Bytes are written until a line terminator (a linefeed or a null character) is encountered. If the terminator is the line feed character it is written out; if the terminator is the null character it is not written out.

system call: _wrseq
number: 15h
purpose: This call writes sequential bytes.

user access: all users

summary:

```
move    <channel>, D1
move.l  <byte count>, D3
lea     <buffer>, A0
jsys    #_wrseq
move.l  D3, <bytes written>
```

calling parameters:

D1 The D1 register contains the channel number of the file.

D3.L The D3.L register contains the number of sequential bytes to be written to the file.

A0 The A0 register points to the buffer where the bytes to be written are stored.

return parameters:

D3.L The D3 register contains the number of bytes actually written.

possible errors:

?notopen
?filaccess
?ioerror

The _wrseq call writes a series of bytes to the file opened on the specified channel. The bytes are written at the current file position. Note that this may overwrite information previously written to the file.

Chapter 3

Z80 CROMIX SYSTEM CALL SUMMARY

The Cromix-Plus Operating System contains a Z80 emulator capable of running Z80 programs, even though the operating system itself runs on the 68000. Consequently, you may wish to write Z80 programs using Z80 Cromix system calls, to be run under Cromix-Plus. The material in this chapter is provided for this purpose.

The Z80 Cromix system calls are nearly identical to the 68000 versions described in the previous chapter. Most operate in exactly the same way. The only difference is the names of the registers which contain the various parameters. In case of an error, the Carry flag is set and the error number is returned in the **a** register.

The following table summarizes the Z80 system calls and the registers they use. For the full description of each call, refer to the previous chapter.

Call	Number	Calling Parameters	Return Parameters
.alarm	43h	hl = number of seconds	

.caccess	27h	b = channel
		c = access bits	

.cchstat	23h	b = channel
		c = status type
		de = new value
			see table 3-1

.chdup	0Ah	b = existing channel	c = duplicate channel

.chkdev	07h	c = type of device
		d = major device number
		e = minor device number	

.clink	25h	b = channel
		de = new pathname	

.close	0Bh	b = channel	

.create	08h	hl = pathname
		c = access mode
		d = exclusive mode
			b = channel

Call	Number	Calling Parameters	Return Parameters

.cstat	21h	de = buffer
		b = channel
		c = desired information
			see table 3-2

.delete	06h	hl = pathname	

.divd	54h	dehl = dividend
		bc = divisor
			hl = quotient
			de = remainder

.error	1Ch	a = error number
		b = channel	

.exchg	0Ch	b = channel number
		c = channel number	

.exec	4Ch	de = argument list
		hl = pathname	

.exit	46h	hl = termination status	

.faccess	26h	c = access bits
		hl = pathname	

.fchstat	22h	c = status type
		de = new value
		hl = pathname
			see table 3-3

.fexec	4Bh	b = signal mask
		c = signal values
		hl = pathname
		de = argument list
			hl = new pid

.flink	24h	de = new pathname
		hl = old pathname	

.fshell	48h	b = signal mask
		c = signal values
		de = argument list
			hl = new pid

.fstat	20h	c = desired information
		hl = pathname
		de = buffer
			see table 3-4

.getdate	30h		d = day of the week
			e = year
			h = month
			l = day of the month

.getdir	02h	hl = buffer	

Call	Number	Calling Parameters	Return Parameters
.getgroup	36h	c = id type	hl = group id

.getmode	12h	b = channel
		c = mode type
			d, de, or dehl = return value

.getpos	10h	b = channel	dehl = file position
.getprior	38h		l = priority number
.getproc	3Ah		hl = process id

.gettime	32h		e = hour
			h = minute
			l = second

.getuser	34h	c = id type	hl = user id
.indirect	51h	a = call number Other registers are used according to call number	

.kill	41h	c = signal type
		hl = process id	

.lock	3Eh	c = lock type
		de = lock length
		hl = lock sequence	

.makdev	00h	c = type of device
		d = major device number
		e = minor device number
		hl = pathname	

.makdir	01h	hl = pathname	

.mount	04h	c = type of access
		hl = dummy pathname
		de = device pathname	

.mult	53h	bc = multiplier
		hl = multiplicand
			dehl = product

.open	09h	c = access mode
		d = exclusive mode
		hl = pathname
			b = channel

.pause	44h		

Call	Number	Calling Parameters	Return Parameters

.pipe	0Eh		b = read channel
			c = write channel

.printf	1Bh	b = channel
		hl = control string
		Arguments on stack	

.rdbyte	16h	b = channel	a = byte

.rdline	18h	b = channel
		de = maximum bytes
		hl = buffer
			de = bytes read

.rdseq	14h	b = channel
		de = maximum bytes
		hl = buffer
			de = bytes read

.setdate	31h	e = year
		h = month
		l = day of the month	

.setdir	03h	hl = pathname	

.setgroup	37h	b = type to change
		c = new id type
		hl = new group id	

.setmode	13h	b = channel
		c = mode type
		d = new value
		e = mask
			d, de, or dehl = old value

.setpos	11h	b = channel
		c = mode
		dehl = file pointer	

.setprior	39h	l = priority number	

.settime	33h	e = hour
		h = minute
		l = second	

.setuser	35h	b = type to change
		c = new id type
		hl = new user id	

.shell	49h	de = argument list	

.signal	40h	c = type of signal
		hl = execution address
			hl = previous address

Call	Number	Calling Parameters	Return Parameters
.sleep	42h	hl = seconds to sleep	hl = seconds left
.trunc	0Dh	b = channel	

.unlock	3Fh	c = lock type
		de = lock length
		hl = lock sequence	

.unmount	05h	c = eject flag
		hl = device pathname	

.update	52h		
.version	55h		hl = version number

.wait	45h	c = conditional flag
		hl = process id
			de = process status
			c = system status
			hl = child pid

.wrbyte	17h	a = byte
		b = channel	

.wrline	19h	b = channel
		hl = buffer
			de = bytes written

.wrseq	15h	b = channel
		de = byte count
		hl = buffer
			de = bytes written

Table 3-1: Z80 CCHSTAT CALLS

Who*	C Register	Status Type	Location of New Information
p	ST.OWNER	owner id	de = new value
p	ST.GROUP	group id	de = new value
p&o	ST.AOWNER	access owner	d = new value, e = mask
p&o	ST.AGROUP	access group	d = new value, e = mask
p&o	ST.AOTHER	access public	d = new value, e = mask
p	ST.TCREATE	time created	de -> 6-byte buffer
p	ST.TMODIFY	time last modified	de -> 6-byte buffer
p	ST.TACCESS	time last accessed	de -> 6-byte buffer
p	ST.TDUMPED	time last dumped	de -> 6-byte buffer

*p = privileged user
o = owner

Table 3-2: Z80 CSTAT CALLS

C Register	Information Returned	Location of Information
ST.ALL	all of inode	de -> 128-byte inode buffer
ST.OWNER	owner id	de
ST.GROUP	group id	de
ST.AOWNER	access owner	d
ST.AGROUP	access group	d
ST.AOTHER	access public	d
ST.FTYPE	file type	d = IS.ORDIN IS.DIRECT IS.CHAR IS.BLOCK
ST.SIZE	file size	dehl
ST.NLINKS	number of links	de
ST.INUM	inode number	de
ST.TCREATE	time created	de -> 6-byte buffer
ST.TMODIFY	time last modified	de -> 6-byte buffer
ST.TACCESS	time last accessed	de -> 6-byte buffer
ST.TDUMPED	time last dumped	de -> 6-byte buffer

ST.DEVNO	device number	d = major device number
		e = minor device number

ST.DEVICE	device number	d = major device number
		e = minor device number

ST.PDEVNO	device number	d = major device number
		e = minor device number

Table 3-3: Z80 FCHSTAT CALLS

Who*	C Register	Status Type	Location of New Information
p	ST.OWNER	owner id	de = new value
p	ST.GROUP	group id	de = new value
p&o	ST.AOWNER	access owner	d = new value, e = mask
p&o	ST.AGROUP	access group	d = new value, e = mask
p&o	ST.AOTHER	access public	d = new value, e = mask
p	ST.TCREATE	time created	de -> 6-byte buffer
p	ST.TMODIFY	time last modified	de -> 6-byte buffer
p	ST.TACCESS	time last accessed	de -> 6-byte buffer
p	ST.TDUMPED	time last dumped	de -> 6-byte buffer

*p = privileged user
o = owner

Table 3-4: Z80 FSTAT CALLS

C Register	Information Returned	Location of Information
ST.ALL	all of inode	de -> 128-byte inode buffer
ST.OWNER	owner id	de
ST.GROUP	group id	de
ST.AOWNER	access owner	d
ST.AGROUP	access group	d
ST.AOTHER	access public	d
ST.FTYPE	file type	d = IS.ORDIN IS.DIRECT IS.CHAR IS.BLOCK
ST.SIZE	file size	dehl
ST.NLINKS	number of links	de
ST.INUM	inode number	de
ST.TCREATE	time created	de -> 6-byte buffer
ST.TMODIFY	time last modified	de -> 6-byte buffer
ST.TACCESS	time last accessed	de -> 6-byte buffer
ST.TDUMPED	time last dumped	de -> 6-byte buffer

ST.DEVNO	device number	d = major device number
		e = minor device number

ST.DEVICE	device number	d = major device number
		e = minor device number

ST.PDEVNO	device number	d = major device number
		e = minor device number

Chapter 4

DISK ALLOCATION UNDER CROMIX-PLUS

This chapter describes disk allocation under the Cromix Operating System. Any small or large floppy disk or hard disk formatted for use under the Cromix system is divided into three major sections: the **System Area, Inode Area,** and **Data Area.** These disks are formatted with a block size of 512 bytes decimal.

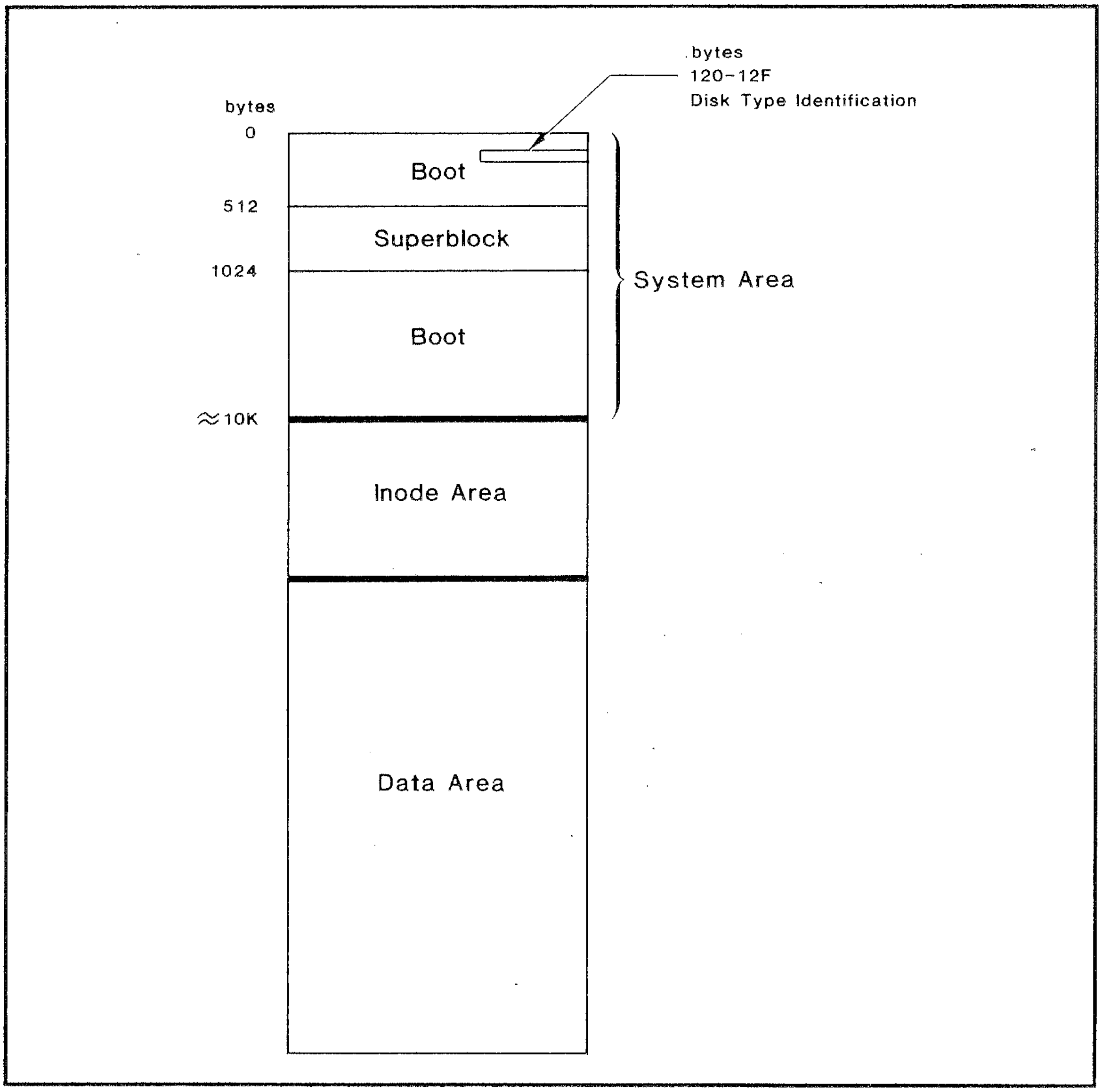

Figure 4-1: LAYOUT OF A CROMIX DISK

SYSTEM AREA

The System Area has a default size of 10K bytes for all disk types. Although it is not recommended, the size of this area can be specified when running the **Makfs** (make file system) utility program.

The System Area contains system information required for booting up (boot tracks) and disk type identification. In addition, it contains the Superblock, and, for hard disks, the alternate track table and the partition table.

Disk Type Identification

On Cromix-format floppy disks, bytes 120 through 127 (in the first block) contain ASCII-encoded data detailing the type and use of the disk.

Floppy disks have six letters in this position. When formatted for use with the Cromix Operating System, byte 120 contains a **C**. Byte 121 contains an **S** or **L**, to indicate a Small (5") or Large (8") floppy disk. Bytes 122-123 contain the characters **SS** or **DS**, indicating a Single Sided or Double Sided Disk. Bytes 124-125 contain the characters **SD** or **DD**, indicating a Single Density or Double Density disk. Bytes 126-127 are not significant, but are reserved for future use.

Cromix-Plus also supports uniform-format floppy disks, which contain no identification information in the first block. In uniform format, all tracks are the same. All sectors are the same size: the sector size might be 128, 256, or 512 bytes.

On hard disks, bytes 68h through 7Fh contain disk type identification. The following table details this area of the disk.

68-69	Number of cylinders, not counting alternate tracks (2 bytes)
6A-6B	Number of alternate tracks (2 bytes)
6C	Number of surfaces (1 byte)
6D	Number of sectors per track (1 byte)
6E-6F	Number of bytes per sector (2 bytes)
70-71	Byte count of start of alternate track table (2 bytes)
72-73	Cylinder number of start of disk (2 bytes)
74-75	Cylinder number where alternate tracks are located (2 bytes)
76-77	Byte count of start of partition table (2 bytes)
78-7B	Hard disk identifier, usually CSTD (4 bytes)
7C-7D	Cylinder number where write precompensation starts
7E-7F	Reserved for future use (4 bytes)

Superblock

The second block (bytes 512-1023) is the Superblock. This block contains housekeeping information for the disk, including the **Block Free List** and the **Inode Free List.**

The Block Free List (sometimes called the Free List) is a stack of 80 4-byte pointers, preceded by a 2-byte counter. Each pointer in the Block Free List points to a disk block not in use. As information is deleted from the disk, the Block Free List grows; as information is written to the disk, it shrinks.

The last pointer used (actually, the first pointer in the list) points to a block on the disk that contains another Block Free List. When the Block Free List in the Superblock is exhausted, the next Block Free List is loaded into the Superblock. When the Block Free List in the Superblock is full, it is moved to the Data Area of the disk.

The Inode Free List is a stack of 80 2-byte inode numbers preceded by a 2-byte counter. Each entry in the Inode Free List is the number of an unused inode. When this stack is exhausted, the Cromix system searches through the inode table and replenishes the stack with the numbers of additional inodes not in use.

Alternate Track Table

The Alternate Track Table for the hard disk is located at the top of the System Area, before the Inode Area.

INODE AREA

An inode is a descriptor for one file; it contains a collection of information pertaining to the file.

The first 48 bytes contain information on the number of links to the file, allowable access modes, and most recent access times for various types of access.

The last 80 bytes of the inode contain 4-byte pointers to the file itself. The first 16 of these pointers each points to a block of the file. The first pointer points to the first block (bytes 0-511); the second pointer points to the second block (bytes 512-1023), and so on. This continues until the whole file has been pointed to, or until the sixteenth pointer has been used (pointing to bytes 7680-8191). Thus, if the file is 8 Kbytes or smaller, only the first 16 (or fewer) pointers need be used.

If the file described by the inode is larger than 8 Kbytes, the seventeenth pointer is used. This pointer points to a block of 128 pointers. Each of these pointers points to a block of the file in a manner similar to the first 16 pointers described above. Thus the seventeenth pointer describes the next 64 Kbytes of the file.

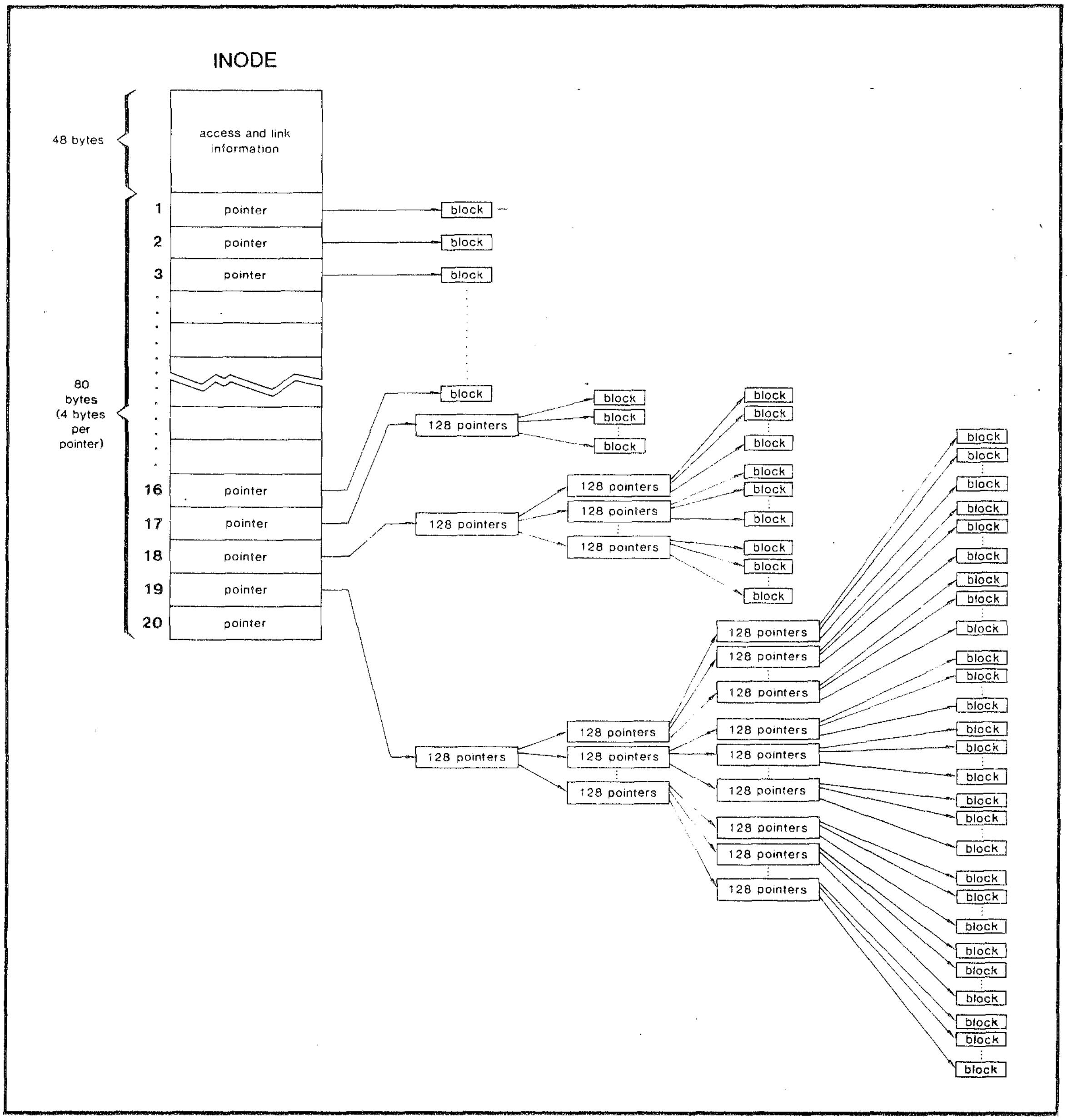

Figure 4-2: INODE LAYOUT

If the file is larger than 72 Kbytes, the eighteenth pointer is used. This pointer points to a block of 128 pointers. Each of these points to a block of 128 pointers. These pointers, in turn, point to a block in the file. Thus, the eighteenth pointer describes the next 8192 Kbytes of the file. The nineteenth pointer extends one more level, covering the next 1,048,576 Kbytes of the file.

DATA AREA

The Data Area occupies most of the disk. All data on the disk is stored in the data area. All blocks pointed to by inodes are in this area.

Appendix A

68000 EQUATE LISTINGS

/EQU/JSYSEQU.H

```
/* Jsysequ.h:  Cromemco C I/O header file

   Copyright (c) 1985 by Cromemco, Inc., All Rights Reserved

   This file contains declarations of all values which are
   used during calls to the Cromix-Plus operating system.

   Oct 25, 1985
*/



/*
   Standard channel numbers
*/

#define STDIN          0      /* Standard input channel         */
#define STDOUT         1      /* Standard output channel        */
#define STDERR         2      /* Standard error channel         */


/*
   Access modes for create
*/

#define op_read        0      /* Read only                      */
#define op_write       1      /* Write only                     */
#define op_rdwr        2      /* Read and write                 */
#define op_append      3      /* Append only                    */
#define op_xread       4      /* Exclusive read only            */
#define op_xwrite      5      /* Exclusive write only           */
#define op_xrdwr       6      /* Exclusive read and write       */
#define op_xappend     7      /* Exclusive append only          */

#define op_truncf      0x80   /* Truncate on create flag        */
#define op_condf       0x40   /* Conditional create flag        */
#define op_force       0x20   /* Force open on block device     */
```

```
/*
   Modes for setpos system call
*/

#define fwd_begin        0     /* Forward from the beginning of the file       */
#define fwd_current      1     /* Forward from the current position            */
#define fwd_end          2     /* Forward from the end of the file             */
#define bak_current     -1     /* Backward from the current file position      */
#define bak_end         -2     /* Backward from the end of the file            */


/*
   Status types for _fstat, _cstat, _fchstat, _cchstat
*/

#define st_all           0     /* All of inode (128 bytes)                 */
#define st_owner         1     /* Owner                                    */
#define st_group         2     /* Group                                    */
#define st_aowner        3     /* Owner access, mask                       */
#define st_agroup        4     /* Group access, mask                       */
#define st_aother        5     /* Other access, mask                       */
#define st_ftype         6     /* File type, special device #              */
#define st_size          7     /* File size                                */
#define st_nlinks        8     /* Number of links                          */
#define st_inum          9     /* Inode number                             */
#define st_device       10     /* Device containing inode                  */
#define st_tcreate      11     /* Time created                             */
#define st_tmodify      12     /* Time last modified                       */
#define st_taccess      13     /* Time last accessed                       */
#define st_tdumped      14     /* Time last dumped                         */
#define st_devno        15     /* Device number if inode is a device       */
#define st_pdevno       16     /* Phys device # if inode is a device       */


/*
   Status types for _ustat, _uchstat
*/

#define usr_ctty         0     /* Controlling tty device number            */
#define usr_prior        1     /* Process priority                         */
#define usr_parent       2     /* Parent process id                        */
#define usr_memp         3     /* Address of user code                     */
#define usr_mems         4     /* Size of code memory                      */
#define usr_time         5     /* Process time in miliseconds              */
#define usr_ctime        6     /* Children time in miliseconds             */
#define usr_user         7     /* Effective user id                        */
#define usr_group        8     /* Effective group id                       */
#define usr_term         9     /* Terminal identification                  */
```

```
/*
   File types for st_ftype
*/

#define is_ordin          0      /* Ordinary file                    */
#define is_direct         1      /* Directory file                   */
#define is_char           2      /* Character device                 */
#define is_block          3      /* Block device                     */
#define is_pipe           4      /* Pipe file                        */


/*
   Mask values for file access flags
*/

#define ac_read           0x01   /* Read access bit                  */
#define ac_exec           0x02   /* Execute access bit               */
#define ac_writ           0x04   /* Write access bit                 */
#define ac_apnd           0x08   /* Append access bit                */


/*
   Id types and values for _setuser, _getuser, _setgroup, _getgroup */

#define id_effective      0      /* Effective id                     */
#define id_login          1      /* Login id                         */
#define id_program        2      /* Program id                       */
#define id_number         3      /* Id contained in idnumber         */


/*
   Signal types
*/

#define sigabort          1      /* Control-C key                    */
#define siguser           2      /* User specifiable key             */
#define sigkill           3      /* Kill signal                      */
#define sigterm           4      /* Terminate                        */
#define sigalarm          5      /* Alarm                            */
#define sigpipe           6      /* Broken pipe signal               */
#define sighangup         7      /* Modem hang up                    */
                                 /* Reserved                         */
```

```
#define SIGABORT        (1 << sigabort - 1)
#define SIGUSER         (1 << siguser - 1)
#define SIGKILL         (1 << sigkill - 1)
#define SIGTERM         (1 << sigterm - 1)
#define SIGALARM        (1 << sigalarm - 1)
#define SIGPIPE         (1 << sigpipe - 1)
#define SIGHANGUP       (1 << sighangup- 1)


/*
   Cromix-Plus System Call Numbers
*/

#define _makdev         0x00  /* Make device entry                        */
#define _makdir         0x01  /* Make a directory                         */
#define _getdir         0x02  /* Get current directory name               */
#define _setdir         0x03  /* Change current directory                 */
#define _mount          0x04  /* Mount file system                        */
#define _unmount        0x05  /* Unmount file system                      */
#define _delete         0x06  /* Delete file                              */
#define _chkdev         0x07  /* Check for device driver                  */
#define _create         0x08  /* Create & open file                       */
#define _open           0x09  /* Open file                                */
#define _chdup          0x0A  /* Duplicate channel                        */
#define _close          0x0B  /* Close file                               */
#define _exchg          0x0C  /* Exchange the contents of two inodes      */
#define _trunc          0x0D  /* Truncate open file                       */
#define _pipe           0x0E  /* Generate a pipe                          */

#define _getpos         0x10  /* Get file position                        */
#define _setpos         0x11  /* Set file position                        */
#define _getmode        0x12  /* Get device characteristics               */
#define _setmode        0x13  /* Set device characteristics               */
#define _rdseq          0x14  /* Read n bytes                             */
#define _wrseq          0x15  /* Write n bytes                            */
#define _rdbyte         0x16  /* Read 1 byte                              */
#define _wrbyte         0x17  /* Write 1 byte                             */
#define _rdline         0x18  /* Read a line                              */
#define _wrline         0x19  /* Write a line                             */

#define _printf         0x1B  /* Print formatted string                   */
#define _error          0x1C  /* Print error message                      */

#define _fstat          0x20  /* Get file status (inode)                  */
#define _cstat          0x21  /* Get channel status (inode)               */
#define _fchstat        0x22  /* Change file status                       */
#define _cchstat        0x23  /* Change channel status                    */
#define _flink          0x24  /* Link to file                             */
#define _clink          0x25  /* Link to open channel                     */
#define _faccess        0x26  /* Test file access                         */
#define _caccess        0x27  /* Test channel access                      */
#define _ustat          0x28  /* Get process table information            */
```

```
#define _uchstat        0x29  /* Change process table information      */

#define _getdate        0x30  /* Get date                              */
#define _setdate        0x31  /* Set date                              */
#define _gettime        0x32  /* Get time                              */
#define _settime        0x33  /* Set time                              */
#define _getuser        0x34  /* Get user id                           */
#define _setuser        0x35  /* Set user id                           */
#define _getgroup       0x36  /* Get group id                          */
#define _setgroup       0x37  /* Set group id                          */
#define _getprior       0x38  /* Get the current process priority      */
#define _setprior       0x39  /* Set the current process priority      */
#define _getproc        0x3A  /* Get process id                        */

#define _ksam           0x3D  /* Ksam system call                      */
#define _lock           0x3E  /* Lock key                              */
#define _unlock         0x3F  /* Unlock key                            */
#define _signal         0x40  /* Set up to receive a signal            */
#define _kill           0x41  /* Send a signal                         */
#define _sleep          0x42  /* Sleep for specified number of secs    */
#define _alarm          0x43  /* Set alarm clock                       */
#define _pause          0x44  /* Pause for alarm clock                 */
#define _wait           0x45  /* Wait for child process                */
#define _exit           0x46  /* Exit process (close files)            */
#define _fork           0x47  /* Fork a process                        */
#define _fshell         0x48  /* Fork a shell process                  */
#define _shell          0x49  /* Transfer to shell process             */

#define _fexec          0x4B  /* Fork and execute program              */
#define _exec           0x4C  /* Execute program                       */
#define _execz80        0x4D  /* Execute z80 program                   */
#define _ptrace         0x4E  /* Debug system call                     */
#define _memory         0x50  /* Allocate user memory                  */
#define _indirect       0x51  /* System call in D0-register            */
#define _update         0x52  /* Update disk I/O buffers               */
#define _mult           0x53  /* Multiply                              */
#define _divd           0x54  /* Divide                                */
#define _version        0x55  /* Get system version #                  */
#define _boot           0x56  /* Boot new operating system             */


/*
   Cromix-Plus error numbers
*/

#define _badchan        1     /* Bad channel #                         */
#define _toomany        2     /* Channel already open                  */
#define _notopen        3     /* Channel not open                      */
#define _endfile        4     /* End-of-file                           */
#define _ioerror        5     /* I/O error                             */
#define _filtable       6     /* File table exhausted                  */
#define _notexist       7     /* File does not exist                   */
#define _badname        8     /* Bad file name                         */
```

```
#define _diraccess      9    /* Directory access                          */
#define _filaccess      10   /* File access                               */
#define _exists         11   /* File already exists                       */
#define _nospace        12   /* No disk space left                        */
#define _noinode        13   /* No inodes left                            */
#define _inotable       14   /* Inode table exhausted                     */
#define _badcall        15   /* Illegal system call                       */
#define _filsize        16   /* File size too big                         */
#define _mnttable       17   /* Mount table exhausted                     */
#define _notdir         18   /* Not a directory                           */
#define _isdir          19   /* Is a directory                            */
#define _priv           20   /* Privileged system call                    */
#define _notblk         21   /* Not a block special device                */
#define _fsbusy         22   /* File system busy                          */
#define _notordin       23   /* Not an ordinary file                      */
#define _notmount       24   /* Device not mounted                        */
#define _nochild        25   /* No child processes                        */
#define _nomemory       26   /* Not enough memory                         */
#define _ovflo          27   /* Divide overflow                           */
#define _argtable       28   /* Argument table exhausted                  */
#define _arglist        29   /* Arg list too big                          */
#define _numlinks       30   /* Too many number of links                  */
#define _difdev         31   /* Cross-device link                         */
#define _nodevice       32   /* No special device                         */
#define _usrtable       33   /* User process table exhausted              */
#define _badvalue       34   /* Value out of range                        */
#define _notconn        35   /* I/O device not connected                  */
#define _devopen        36   /* Device open error                         */
#define _diruse         37   /* Directory in use (delete)                 */
#define _filuse         38   /* File in use (exclusive access)            */
#define _nomatch        39   /* No match on ambiguous name                */
#define _chnaccess      40   /* Channel access                            */
#define _notcromix      41   /* Not a cromix disk                         */
#define _badfree        42   /* Bad free list                             */
#define _badinum        43   /* Bad inode number                          */
#define _readonly       44   /* Device mounted for read only              */
#define _noproc         45   /* Process does not exist                    */
#define _ssignal        46   /* System call was aborted                   */
#define _badpipe        47   /* Bad call on pipe                          */
#define _locked         48   /* Locked                                    */
#define _deadlock       49   /* Deadlocked                                */
#define _lcktable       50   /* Lock table exhausted                      */
#define _tapeio         51   /* Tape I/O error                            */
#define _badio          52   /* I/O error                                 */
#define _not68000       53   /* 68000 programs cannot run under Z80       */
#define _badformat      54   /* Bad file format                           */
#define _runaway        55   /* Runaway program aborted                   */
#define _cdossim        56   /* CDOS simulator required                   */
#define _corrupt        57   /* System image corrupted                    */
```

/EQU/MODEEQU.H

```
/*      Modeequ.h:      Cromemco 68000 C I/O header file
        Copyright (c) 1985 by Cromemco, Inc., All Rights Reserved

        This file contains declarations of all values which are
        used in the getmode and setmode Cromix system calls.

        Sep 09-85
*/


#define MD_ISPEED       0               /* input speed                          */
#define MD_OSPEED       1               /* output speed                         */
#define MD_MODE1        2               /* flags:  RAW, ECHO, etc.              */
#define MD_MODED        3               /* delays for NL, CR, etc.              */
#define MD_MODE2        4               /* flags:  PAUSE, XFF, etc.             */
#define MD_MODE3        5               /* flags:  ESCRETN                      */
#define MD_ERASE        6               /* auxiliary erase character            */
#define MD_DELECHO      7               /* erasure echo character               */
#define MD_LKILL        8               /* line kill character                  */
#define MD_USIGNAL      9               /* user signal key                      */
#define MD_LENGTH      10               /* page length (lines)                  */
#define MD_WIDTH       11               /* page width (columns)                 */
#define MD_BMARGIN     12               /* bottom margin (lines)                */
#define MODELEN        (MD_BMARGIN + 1)

#define MD_FORMS       254              /* printer forms number                 */
#define MD_IDENT       255              /* device identification                */


/* the following are for SLPT only */

#define SLPT_BSIZE     MD_ERASE         /* ETX/ACK block size                   */

/* the following are for TYP only */

#define TYP_CWIDTH     64               /* character width in 1/120 in          */
#define TYP_LHEIGHT    65               /* line height in 1/48 in               */
#define TYP_LMARGIN    66               /* left margin in columns (1/10)        */

/* the following are commands, not displacements in the device structure */

#define MD_STATUS      156              /* flag:  character is in one           */
                                        /*        of the input queues           */
#define MD_IFLUSH      155              /* flush input queues                   */
#define MD_FNKEYS      152              /* turn function keys on or off         */
#define MD_PSIGHUP     151              /* signal current process if hang up    */
#define MD_MODEM       148              /* (QTTY and MTTYs only)                */
#define MD_TYP         147              /* (TYPs only)                          */

/* contents of D3-register for MD_ISPEED calls to change the baudrate           */
```

```
#define S_HANGUP        0     /* hang up dataphone     */
/*                      1         50 baud              */
/*                      2         75 baud              */
#define S_110           3     /* 110 baud              */
/*                      4         134.5 baud           */
#define S_150           5     /* 150 baud              */
/*                      6         200 baud             */
#define S_300           7     /* 300 baud              */
/*                      8         600 baud             */
#define S_1200          9     /* 1200 baud             */
/*                     10         1800 baud            */
#define S_2400         11     /* 2400 baud             */
#define S_4800         12     /* 4800 baud             */
#define S_9600         13     /* 9600 baud             */
/*                     14         External A           */
/*                     15         External B           */
#define S_19200        16     /* 19200 baud            */
#define S_CTSWAIT     125     /* wait for clear to send            */
#define S_NOCHG       126     /* no change of baudrate             */
#define S_UNINIT      127     /* uninitialized baudrate            */
#define Sfl_AUTO      0x80    /* (bit 7): input CRs from keyboard to      */
                              /*       set baud                           */


/* contents of the D3-register & D4-register for MD_MODE1 calls           */

#define TANDEM        0x01
#define XTAB          0x02    /* expand TABs                              */
#define LCASE         0x04    /* convert alphabetics to lower case        */
#define ECHO          0x08    /* echo input                               */
#define CRDEVICE      0x10    /* on input, map CR into NL                 */
                              /* on output, echo LF or CR as CRLF         */
#define RAW           0x20    /* on input, return after each              */
                              /* character                                */
                              /* and treat ^C, ^S, ^Q as regular          */
                              /* input                                    */
#define ODD           0x40    /* parity function bits                     */
#define EVEN          0x80

/* contents of the D3-register & D4-register for MD_MODED calls           */

#define NLDELAY       0x03    /* (pairs of bits)                          */
#define TABDELAY      0x0C
#define CRDELAY       0x30
#define FFDELAY       0x40    /* (single bits)                            */
#define BSDELAY       0x80

/* contents of the D3-register & D4-register for MD_MODE2 calls           */

#define PAUSE         0x01    /* wait for CNTRL-Q after a page            */
                              /*           is output                      */
#define NOTIMMECHO    0x02    /* do not echo characters                   */
                              /*           typed-ahead                    */
```

```
#define NOECNL          0x04    /* do not echo NLs                       */
#define SGENABLE        0x08    /* user-specifiable key signal enable    */
#define ABENABLE        0x10    /* CNTRL-C key signal enable             */
#define XFF             0x20    /* expand FFs                            */
#define WRAP            0x40    /* wrap-around if page width is exceeded */
#define SIGALLC         0x80    /* send siguser signal for each key pushed */

/* contents of the D3-register & D4-register for MD_MODE3 calls          */

#define ESCRETN         0x01    /* ESC causes input line to be           */
                                /*          returned                     */
#define FNKEYS          0x02    /* enable response to 3102 function keys */
#define HUPENAB         0x04    /* hang up modem when device finally closed */
#define SIGHUPALL       0x08    /* send sighangup signals to all processes */
                                /* which use this tty if modem hangs up    */
#define CBREAK          0x10    /* on input, return after each character, */
                                /* no erase, linekill, or eof characters  */
#define BINARY          0x20    /* on input, return after each           */
                                /* character, no erase, linekill, or     */
                                /* eof characters, no output pause or    */
                                /* output width truncation, treat x-off,*/
                                /* x-on as regular input, no tandem mode*/
                                /* (ie, no input buff ctl), no abort     */
                                /* signal (^C), no user signal, no       */
                                /* changing or checking parity bit, no   */
                                /* delays after control chars as nls,    */
                                /* no echoing, no character              */
                                /* transformations, no function key      */
                                /* decoding.                             */
#define CRIGNORE        0x40    /* On output, ignore CR and change LF    */
                                /* to CR                                 */
#define DISCARD         0x80    /* discard the device when it is no      */
                                /*          longer open                  */

/* bits of the D3-register for MD_STATUS calls                           */

#define INOTEMPTY       0x01    /* there is a character in the input     */
                                /* buffer (but if not RAW mode, it won't*/
                                /* be accessible until a whole line is   */
                                /* entered)                              */

/* contents of the D3-register for MD_MODEM _getmode call                */

#define RXDA            0x01    /* Receiver Data Available               */
#define TXBE            0x04    /* Transmitter Buffer Empty              */
#define DCD             0x08    /* Data Carrier Detect                   */
#define CTS             0x20    /* Clear To Send                         */
#define RXBREAK         0x80    /* Receiver data line broken             */

/* contents of the D4-register for MD_MODEM _getmode call                */

#define notRI           0x40    /* Not Ringing                           */
#define notDSR          0x80    /* Data Set not Ready                    */
```

```
/* contents of the D3-register & D4-register for MD_MODEM _setmode call   */

#define RTS             0x02    /* Request To Send                        */
#define TXBREAK         0x10    /* Break the Transmitter line             */
#define DTR             0x80    /* Data Terminal Ready                    */

/* contents of the D3-register for MD_TYP call                            */

#define TYPCHK          0x02    /* The 3355 printer is in check cond.     */
#define TYPPAP          0x04    /* The 3355 printer is out of paper       */
#define TYPRIB          0x08    /* The 3355 printer is out of ribbon      */
#define TYPOFL          0x10    /* The 3355 printer is off-line           */

/* contents of D3-register for MD_IDENT call                              */

#define ID_TTY           0      /* Tuart terminal                         */
#define ID_QTTY          1      /* Quadart or Octart terminal             */
#define ID_LPT           2      /* Parallel printer                       */
#define ID_TYP           3      /* Fully formed printer                   */
#define ID_SLPT          4      /* Serial printer                         */
#define ID_QSLPT         5      /* Serial printer on quadart              */
#define ID_CNET          6      /* CNET driver                            */
#define ID_FFP           7      /* FFP processor driver                   */
#define ID_SYSTEM        8      /* System device                          */
#define ID_TIMER         9      /* Timer device                           */
#define ID_TAPE         10      /* Half inch tape drive                   */
#define ID_SCC          11      /* SCC terminal                           */
                                /* Values 12 .. 127 reserved              */
                                /* Values 128 .. 255 reserved for user    */
                                /* defined drivers and devices            */
```

/EQU/BMODEEQU.H

```
/*
        Mode definitions for block devices

        Cromemco Inc.
        Aug 24, 1985
*/


/*
        Mode numbers for getmode and setmode calls
*/

#define BMD_STATUS      0       /* Get/set status byte                          */
#define BMD_FLG1        1       /* Get/set flag1 byte                           */
#define BMD_FLG2        2       /* Get/set flag2 byte                           */
#define BMD_FLG3        3       /* Get/set flag3 byte                           */
#define BMD_SIZE        4       /* Get number of bytes on device                */
#define BMD_SEEK        5       /* Seek                                         */
#define BMD_INIT        6       /* Initialize track                             */
#define BMD_PRDWRT      7       /* Primitive read/write                         */
#define BMD_RDWRT       8       /* Special read/write                           */
#define BMD_RPM         9       /* Get RPM                                      */
#define BMD_VERSION     10      /* Version number                               */
#define BMD_PHYCHAR     11      /* Physical Characteristics                     */
#define BMD_LDFIRM      12      /* Load firmware                                */
#define BMD_SOFT        13      /* Accumulated number of retries                */
#define BMD_HARD        14      /* Accumulated number of hard errors            */
#define BMD_RETRY       15      /* Number of retries before hard error          */
                                /* Values 16 .. 63 reserved                     */
                                /* Values 64 .. 127 special device modes      */
                                /* Values 128 .. 255 reserved for user          */
                                /* supplied drivers                             */


/*
        Floppy tape special numbers
*/

#define BMD_RETEN       64      /* Number of tape repositions before            */
                                /*      a retension                             */

/*
        IMI disk special numbers
*/

#define BMD_IMITYPE     64      /* Get type of IMI drive                        */

/*
```

```
        Values returned by BMD_IMITYPE
*/

#define IM_50070        0x01    /* IMI Model 50070                      */
#define IM_5007W        0x02    /* IMI Model 5007W                      */
#define IM_5018H        0x03    /* IMI Model 5018H                      */
#define IM_7710A        0x04    /* IMI Model 7710A                      */
#define IM_7710B        0x05    /* IMI Model 7710B                      */


/*
        Memory driver special numbers
*/

#define BMD_TMEM        64      /* Total memory                         */
#define BMD_SMEM        65      /* System memory                        */
#define BMD_FMEM        66      /* Free memory                          */
#define BMD_MMEM        67      /* Maximal free memory                  */
#define BMD_CACR        68      /* CACR register                        */


/*
        Mode values and masks for BMD_STATUS calls
*/

#define DS_BUSY         0x01    /* Device Busy (in use)                 */
#define DS_WANT         0x02    /* Device Wanted (do wakeup)            */
#define DS_READ         0x04    /* Read-only device                     */
#define DS_MODF         0x08    /* Super-block modified                 */
#define DS_MOUNT        0x10    /* Device mounted                       */
#define DS_HOME         0x20    /* Device has been homed                */
#define DS_BFSTEP       0x40    /* Buffered step flag                   */
#define DS_VERIFY       0x80    /* Verify after write                   */


/*
        Mode values and masks for BMD_FLG1 calls
*/

#define DF_SMALL        0x01    /* 1=small floppy 0=large floppy        */
#define DF_DSIDE        0x02    /* Double sided                         */
#define DF_DDENS        0x04    /* Double density                       */
#define DF_DTRACK       0x08    /* Double tracked                       */
#define DF_CROMIX       0x10    /* Cromix format disk                   */
#define DF_CDOS         0x20    /* Cdos format disk                     */
#define DF_BACKUP       0x40    /* Backup format disk                   */
#define DF_VOICE        0x80    /* 0=step 1=voice coil                  */

/*
        Mode values for BMD_FLG2 calls
*/
```

```
#define D2_SMALL        0       /* Small floppy                         */
#define D2_LARGE        1       /* Large floppy                         */
#define D2_STDC         2       /* STDC Hard disk                       */
#define D2_FSMD         3       /* Fixed part of SMD hard disk          */
#define D2_RSMD         4       /* Removable part of SMD hard disk      */
#define D2_UNIFORM      5       /* Uniform floppy                       */
#define D2_MEMORY       6       /* Processor memory                     */
#define D2_RAM          7       /* RAM disk                             */
#define D2_FTAPE        8       /* Floppy tape                          */
#define D2_HD           9       /* WDI hard disk                        */
                                /* Values 10 .. 127 reserved            */
                                /* Values 128 .. 255 reserved for user  */
                                /* supplied drivers                     */

/*
        Mode values and masks for BMD_FLG3 calls
*/

#define D3_WRTPRO       0x01    /* Device is write protected            */
#define D3_INTRPT       0x02    /* Device interrupts                    */
#define D3_DUAL         0x04    /* Dual drive                           */


/*
        Floppy minor device number definition
*/

#define FDENSITY        0x40            /* 0=double density             */
#define FSIDES          0x20            /* 0=double sided               */
#define FDUAL           0x10            /* 1=dual drive (PERSCI)        */
#define FDTRACK         0x08            /* 1=double tracked             */
#define FSIZE           0x04            /* 0=8" 1=5"                    */
#define FUNIT           0x03            /* physical unit number mask    */


/*
        SMD minor device number assignment
*/

#define CONTROLLER      0x80            /* Controller mask              */
#define DRIVE           0x40            /* Drive number mask            */
#define FIXED           0x20            /* Fixed flag mask              */
#define PARTITION       0x1f            /* Partition number             */


/*
        Data structure for BMD_INIT call
*/


typedef struct {
        unsigned short  flags;          /* flags (FDENSITY for floppy)  */
        unsigned short  side;           /* side to be initialized       */
```

```
                unsigned short     track;            /* track to be initialized          */
                unsigned char      *buf;             /* pointer to track image           */
        } bm_init;

        /*
                Data structure for BMD_SEEK call
        */

        typedef struct {
                unsigned char      status;           /* Return status                    */
                unsigned char      ferror;           /* Fatal error number               */
                unsigned char      serror;           /* System error number              */
                unsigned char      verify;           /* Verify seek flag                 */
                unsigned short     side;             /* side                             */
                unsigned short     track;            /* track                            */
        } bm_seek;

        /*
                Data structure for BMD_RDWRT call
        */

        typedef struct {
                unsigned short     read;             /* Read/write flag                  */
                unsigned char      *buf;             /* buffer pointer                   */
                unsigned long      number;           /* no. of blocks to read/write      */
                unsigned long      blknr;            /* starting block number            */
        } bm_rdwrt;

        /*
                Data structure for BMD_PRDWRT call
        */

        typedef struct {
                unsigned char      status;           /* return status                    */
                unsigned char      ferror;           /* fatal error number               */
                unsigned char      serror;           /* system error number              */
                unsigned char      read;             /* Read/write flag                  */
                unsigned char      *buf;             /* buffer pointer                   */
                unsigned short     number;           /* number of sectors to do          */
                unsigned short     sector;           /* starting sector number           */
                unsigned short     surface;          /* surface number to read/write     */
                unsigned short     cylinder;         /* cylinder number to read/write   */
        } bm_prdwrt;

        /*
                Status bits primitive operations
        */

        /*      STDC    */
        #define STS_IOERROR        0x01              /* I/O error                        */
        #define STS_NIOERROR       0x02              /* Non I/O error                    */
        #define STS_SELECT         0x04              /* Error on select                  */
        #define STS_SEEK           0x08              /* Error on seek                    */
```

```
#define STS_PRD         0x10            /* Error on primitive read          */
#define STS_PWR         0x20            /* Error on primitive write         */
#define STS_PTX         0x40            /* Error on transfer                */

/*      Floppy  */

#define FLS_SELECT      0x01            /* Error on select                  */
#define FLS_HOME        0x02            /* Error on home                    */
#define FLS_RDADD       0x03            /* Error on read address            */
#define FLS_SEEK        0x04            /* Error on seek                    */
#define FLS_PREAD       0x05            /* Error on preread                 */
#define FLS_READ        0x06            /* Error on read                    */
#define FLS_WRITE       0x07            /* Error on write                   */
#define FLS_WTRK        0x08            /* Error on write track             */

/* SMD      */

#define SMS_SELECT      0x01            /* Error on select                  */
#define SMS_HOME        0x02            /* Error on home                    */
#define SMS_SEEK        0x03            /* Error on seek                    */
#define SMS_READ        0x04            /* Error on read                    */
#define SMS_WRITE       0x05            /* Error on write                   */
#define SMS_HEAD        0x06            /* Error on select head             */
#define SMS_PREAD       0x07            /* Error on preread                 */

/*
        Data structure for BMD_PHYCHAR call
*/

typedef struct {
        unsigned short  surface;        /* number of surfaces on device     */
        unsigned short  cylinder;       /* number of cylinders on device    */
        unsigned short  sector;         /* number of sectors/track          */
        unsigned short secsiz;          /* number of bytes/sector           */
} bm_phy;

/*
        Data structure for BMD_LDFIRM call
*/

typedef struct {
        unsigned short  flags;          /* flags (see below)                */
        unsigned short  count;          /* number of bytes                  */
        unsigned char   *buf;           /* pointer to firmware              */
} bm_ldfrm;

/*
        Flags
*/

#define LDFRM_DEBUG     0x8d            /* Load debugger firmware           */
#define LDFRM_FIRM      0x8f            /* Load Regular firmware            */
```

/EQU/TMODEEQU.H

```
;       Modeequ.h:      Cromemco 68000 C I/O header file
;       Copyright (c) 1984 by Cromemco, Inc., All Rights Reserved
;
;       This file contains declarations of all values which are
;       used in the getmode and setmode Cromix system calls, for
;       TP tape devices.
;
;       Dec-18-84
;

TPABORT         equ     196     ; re-initialize tape driver
TPFMARK         equ     198     ; write file mark
TPSECURE        equ     199     ; security erase
TPREWIND        equ     200     ; rewind
TPUNLOAD        equ     201     ; rewind and unload
TPMODE          equ     202     ; mode bits
TPFILNO         equ     203     ; file number
TPBLKNO         equ     204     ; block number
TPOBLKLN        equ     205     ; block length for next block written
TPIBLKLN        equ     206     ; block length of first block read
TPOBLKS         equ     207     ; number of blocks written
TPSTAT          equ     208     ; get error (status-2, status-1)

;       TPMODE bits

EOFCLOSE        equ     7       ; write EOF to tape when device closes

;       TPSTAT status bits  (obtained from PIO input port A)
;       These bits are returned in e-register
;       Old names are without leading TP

TPDRVBUSY       equ     7       ; drive busy
TPWRRDY         equ     6       ; FIFO ready for input (used for write)
TPRDRDY         equ     5       ; FIFO output ready (used for read)
TPLOADPT        equ     4       ; load point
TPFBUSY         equ     3       ; formatter busy
TPONLINE        equ     2       ; on line
TPIDENT         equ     1       ; ident
TPRDY           equ     0       ; ready

;       TPSTAT status bits  (obtained from PIO input port B)
;       These bits are returned in e-register
;       Old names are without leading TP

TPHISPEED       equ     7       ; high speed status
TPHARDERR       equ     5       ; hard error
TPFLMARK        equ     4       ; file mark
TPCORERR        equ     3       ; correctable error
TPWRPROT        equ     2       ; file write-protected
TPEOT           equ     1       ; end of tape
TPRWINDING      equ     0       ; rewinding
```

/EQU/PTRACE.H

```
/*
        Ptrace information
        EZ -- Jul 29, 1984
*/


typedef struct  _ptc {
        unsigned long   us_D[8];        /* User data registers      */
        unsigned char   *us_A[8];       /* User address registers   */
        unsigned short  us_SR;          /* User status register     */
        unsigned short  *us_PC;         /* User PC register         */
        unsigned short  us_pstat;       /* ptrace status            */
        unsigned short  us_signo;       /* user signal number       */
        short           us_tstat;       /* termination status       */
} ptc;

/*
        Ptrace commands
*/

#define P_START 0                       /* Next fexec is debugged   */
#define P_RDSEQ 1                       /* Read child memory        */
#define P_WRSEQ 2                       /* Write child memory       */
#define P_RDSTA 3                       /* Read child status        */
#define P_WRSTA 4                       /* Write child status       */
#define P_RUN   5                       /* Run child process        */
#define P_TRACE 6                       /* Trace child process      */
#define P_TERM  7                       /*  Terminate child process */

/*
        us_pstat values
*/

#define PS_RUNNING      0               /* Child running, parent asleep */
#define PS_START        1               /* Initial state            */
#define PS_BREAK        2               /* Trap #5 exception        */
#define PS_TRACE        3               /* Trace exception          */
#define PS_SIGNAL       4               /* Program aborted by signal */
#define PS_EXIT         5               /* Program terminated       */
```

Appendix B

Z80 EQUATE LISTINGS

/EQU/JSYSEQU.Z80

```
          list      off,noxref

;
; Cromemco Inc.
; July 9, 1985
;
stdin           equ     0       ; standard input channel
stdout          equ     1       ; standard output channel
stderr          equ     2       ; standard error channel

argc            equ     40H     ; location for argument count
argv            equ     42H     ; location for argument list vector
arg0            equ     0       ; arg offset
arg1            equ     2       ; arg offset
arg2            equ     4       ; arg offset
arg3            equ     6       ; arg offset
arg4            equ     8       ; arg offset

;  C-register modes for .create, .open
;
op.read         equ     0       ; read only
op.write        equ     1       ; write only
op.rdwr         equ     2       ; read and write
op.append       equ     3       ; append only
op.xread        equ     4       ; exclusive read only
op.xwrite       equ     5       ; exclusive write only
op.xrdwr        equ     6       ; exclusive read and write
op.xappend      equ     7       ; exclusive append only

op.truncf       equ     80H     ; truncate on create flag
op.condf        equ     40H     ; conditional create flag
op.force        equ     20H     ; force open of block device

;  C-register file position modes for .setpos
;
fwd.begin       equ     0       ; forward from the beginning of the file
fwd.current     equ     1       ; forward from the current file position
fwd.end         equ     2       ; forward from the end of the file
bak.current     equ     -1      ; backward from the current file position
bak.end         equ     -2      ; backward from the end of the file
```

```
; C-register modes for .fstat, .cstat, .fchstat, .cchstat
;
st.all          equ     0       ; all of inode (128 bytes)
st.owner        equ     1       ; de = owner
st.group        equ     2       ; de = group
st.aowner       equ     3       ; d = owner access, e = mask
st.agroup       equ     4       ; d = group access, e = mask
st.aother       equ     5       ; d = other access, e = mask
st.ftype        equ     6       ; d = file type
st.size         equ     7       ; dehl = file size
st.nlinks       equ     8       ; de = number of links
st.inum         equ     9       ; de = inode number
st.device       equ     10      ; de = device number of file system containing inode
st.tcreate      equ     11      ; de-> time created
st.tmodify      equ     12      ; de-> time last modified
st.taccess      equ     13      ; de-> time last accessed
st.tdumped      equ     14      ; de-> time last dumped
st.devno        equ     15      ; de = device number if inode is a device
st.pdevno       equ     16      ; de = physical device number if inode is a device

; File types for st.ftype
;
is.ordin        defl    0       ; ordinary file
is.direct       defl    1       ; directory file
is.char         defl    2       ; character device
is.block        defl    3       ; block device
is.pipe         defl    4       ; pipe file

; Access bits for access flags
;
ac.read         defl    0       ; read access bit
ac.exec         defl    1       ; execute access bit
ac.writ         defl    2       ; write access bit
ac.apnd         defl    3       ; append access bit

; C-register modes for .setuser, .getuser, .setgroup, .getgroup
;
id.effective    equ     0       ; effective id
id.login        equ     1       ; login id
id.program      equ     2       ; program id
id.hl           equ     3       ; id contained in HL register

; Signals
;
sigabort        defv    1       ; CONTROL-C key
siguser         defv    2       ; user-specifiable key
sigkill         defv    3       ; kill
sigterm         defv    4       ; terminate (catchable)
sigalarm        defv    5       ; alarm clock
sigpipe         defv    6       ; broken pipe
sighangup       defv    7       ; modem hang up
;               defv    8       ; reserved
; System Call Numbers
;
.makdev         equ     00H     ; makdev(d,e,hl)--make device entry
.makdir         equ     01H     ; makdir(hl)--make a directory
.getdir         equ     02H     ; getdir(hl)--get current directory name
.setdir         equ     03H     ; setdir(hl)--change current directory
.mount          equ     04H     ; mount(c,de,hl)--mount file system
.unmount        equ     05H     ; unmount(hl)--unmount file system
.delete         equ     06H     ; delete(hl)--delete file
.chkdev         equ     07H     ; chkdev(d,e)--check for device driver
.create         equ     08H     ; b=create(c,hl)--create & open file
.open           equ     09H     ; b=open(c,hl)--open file
.chdup          equ     0AH     ; c=chdup(b)--duplicate channel
.close          equ     0BH     ; close(b)--close file
.exchg          equ     0CH     ; exchg(b,c)--exchange data in files
.trunc          equ     0DH     ; trunc(b)--truncate open file
.pipe           equ     0EH     ; b,c=pipe()--create a pipe
;               equ     0FH
.getpos         equ     10H     ; dehl=getpos(b)--get file position
.setpos         equ     11H     ; setpos(c,dehl)--set file position
.getmode        equ     12H     ; d=getmode(b,c)--get device characteristics
.setmode        equ     13H     ; d=setmode(b,c,d,e)--set device characteristics
.rdseq          equ     14H     ; de=rdseq(b,de,hl)--read n bytes
.wrseq          equ     15H     ; de=wrseq(b,de,hl)--write n bytes
.rdbyte         equ     16H     ; a=rdbyte(b)--read 1 byte
.wrbyte         equ     17H     ; wrbyte(b,a)--write 1 byte
.rdline         equ     18H     ; de=rdline(b,de,hl)--read a line
.wrline         equ     19H     ; de=wrline(b,hl)--write a line
;               equ     1AH
.printf         equ     1BH     ; printf(b,hl)--print formatted string
.error          equ     1CH     ; error(a,b,de,hl)--print error message
.fstat          equ     20H     ; fstat(c,de,hl)--get file status (inode)
.cstat          equ     21H     ; cstat(b,c,de)--get channel status (inode)
.fchstat        equ     22H     ; fchstat(c,de,hl)--change file status
.cchstat        equ     23H     ; cchstat(b,c,de)--change channel status
.flink          equ     24H     ; flink(de,hl)--link to file
.clink          equ     25H     ; clink(b,de)--link to open channel
.faccess        equ     26H     ; faccess(c,hl)--test file access
.caccess        equ     27H     ; caccess(b,c)--test channel access
;               equ     28H
;               equ     29H
.getdate        equ     30H     ; d,e,h,l=getdate()--get date
.setdate        equ     31H     ; setdate(e,h,l)--set date
.gettime        equ     32H     ; e,h,l=gettime()--get time
.settime        equ     33H     ; settime(e,h,l)--set time
.getuser        equ     34H     ; de,hl=getuser()--get user id
.setuser        equ     35H     ; setuser(hl)--set user id
.getgroup       equ     36H     ; de,hl=getgroup()--get group id
.setgroup       equ     37H     ; setgroup(hl)--set group id
.getprior       equ     38H     ; l=getprior()--get process priority
.setprior       equ     39H     ; setprior(l)--set process priority
.getproc        equ     3AH     ; hl=getproc()--get process id
;               equ     3BH
;               equu    3CH
.ksam           equ     3DH     ; ksam(c,de,hl)--ksam call
.lock           equ     3EH     ; lock(c,de,hl)--lock key
.unlock         equ     3FH     ; unlock(c,de,hl)--unlock key
.signal         equ     40H     ; signal(c,hl)--set up to receive a signal
.kill           equ     41H     ; kill(c,hl)--send a signal
.sleep          equ     42H     ; sleep(hl)--sleep for hl seconds
.alarm          equ     43H     ; alarm(hl)--set alarm clock
.pause          equ     44H     ; pause()--pause for alarm clock
.wait           equ     45H     ; c,de,hl=wait()--wait for child process
.exit           equ     46H     ; exit(hl)--exit process (close files)
; .fork         equ     47H     fork reentrant process
.fshell         equ     48H     ; fshell(de)--fork a shell process
.shell          equ     49H     ; shell(de)--transfer to shell process
;               equ     4AH
.fexec          equ     4BH     ; fexec(bc,de,hl)--fork and execute program
.exec           equ     4CH     ; exec(bc,de,hl)--execute program
; .execz80      equ     4DH     execute z80 program
;               equ     4EH
;               equ     4FH
; .memory       equ     50H     allocate memory
.indirect       equ     51H     ; indirect(a,b,c,de,hl)--system call in A-register
.update         equ     52H     ; update()--update disk I/O buffers
.mult           equ     53H     ; dehl=mult(bc,hl)--multiply
.divd           equ     54H     ; de,hl=divd(dehl,bc)--divide
.version        equ     55H     ; hl=version()--get system version #
.boot           equ     56H     ; boot(hl,de)--boot new operating system
        form

;  Error code definitions
;
?badchan        defv    1       ; bad channel #
?toomany        defv    2       ; channel already open
?notopen        defv    3       ; channel not open
?endfile        defv    4       ; end-of-file
?ioerror        defv    5       ; I/O error
?filtable       defv    6       ; file table exhausted
?notexist       defv    7       ; file does not exist
?badname        defv    8       ; bad file name
?diraccess      defv    9       ; directory access
?filaccess      defv    10      ; file access
?exists         defv    11      ; file already exists
?nospace        defv    12      ; no disk space left
?noinode        defv    13      ; no inodes left
?inotable       defv    14      ; inode table exhausted
?badcall        defv    15      ; illegal system call
?filsize        defv    16      ; file size too big
?mnttable       defv    17      ; mount table exhausted
?notdir         defv    18      ; not a directory
?isdir          defv    19      ; is a directory
?priv           defv    20      ; privileged system call
?notblk         defv    21      ; not a block special device
?fsbusy         defv    22      ; file system busy
```

```
?notordin       defv    23        ; not an ordinary file
?notmount       defv    24        ; device not mounted
?nochild        defv    25        ; no child processes
?nomemory       defv    26        ; not enough memory
?ovflo          defv    27        ; divide overflow
?argtable       defv    28        ; argument table exhausted
?arglist        defv    29        ; bad argument list
?numlinks       defv    30        ; too many links
?difdev         defv    31        ; cross-device link
?nodevice       defv    32        ; no special device
?usrtable       defv    33        ; user process table exhausted
?badvalue       defv    34        ; value out of range
?notconn        defv    35        ; I/O device not connected
?devopen        defv    36        ; device open error
?diruse         defv    37        ; directory in use (delete)
?filuse         defv    38        ; file in use (exclusive access)
?nomatch        defv    39        ; no match on ambiguous name
?chnaccess      defv    40        ; channel access
?notcromix      defv    41        ; not a cromix disk
?badfree        defv    42        ; bad free list
?badinum        defv    43        ; bad inode number
?readonly       defv    44        ; device mounted for read only
?noproc         defv    45        ; process does not exist
?signal         defv    46        ; system call was aborted
?badpipe        defv    47        ; bad call on a pipe
?locked         defv    48        ; locked
?deadlock       defv    49        ; deadlocked
?lcktable       defv    50        ; lock table exhausted
?tapeio         defv    51        ; tape I/O error
?badio          defv    52        ; bad I/O
?not68000       defv    53        ; 68000 programs cannot run under Z80
?badformat      defv    54        ; bad file format
?runaway        defv    55        ; runaway program aborted
?cdossim        defv    56        ; CDOS simulator required
?corrupt        defv    57        ; system image corrupted

        list    on,xref
```

/EQU/MODEEQU.Z80

```
        list    off
        list    noxref  ; (use this line only with ASMB version 3.08 or later)

;
; Cromemco Inc.
; September 9, 1985
;
; Mode definitions for terminals and printers,
; TTY, QTTY, MTTY, LPT, SLPT, QSLPT, and TYP

; C-register values for .GETMODE and .SETMODE system calls
;
MD_ISPEED       defv    0       ; input speed
MD_OSPEED       defv    1       ; output speed
MD_MODE1        defv    2       ; flags: RAW, ECHO, etc.
MD_MODED        defv    3       ; delays for NL, CR, etc.
MD_MODE2        defv    4       ; flags: PAUSE, XFF, etc.
MD_MODE3        defv    5       ; flags: CBREAK, VRAW, etc.
MD_ERASE        defv    6       ; auxiliary erase character
MD_DELECHO      defv    7       ; erasure echo character
MD_LKILL        defv    8       ; line kill character
MD_USIGNAL      defv    9       ; SIGUSER signal key
MD_LENGTH       defv    10      ; page length (lines)
MD_WIDTH        defv    11      ; page width (columns)
MD_BMARGIN      defv    12      ; bottom margin (lines)
MODELEN         defv    MD_BMARGIN + 1
MD_FORMS        defv    254     ; printer forms number
MD_IDENT        defv    255     ; device identification

; More c-register values for SLPT only
SLPT_BSIZE      defv    MD_ERASE        ;ETX/ACK block size

; More c-register values for TYP only
;
TYP_CWIDTH      defv    64      ; character width in 1/120 inches
TYP_LHEIGHT     defv    65      ; line height in 1/48 inches
TYP_LMARGIN     defv    66      ; left margin in columns (1/10 inches)

; More c-register values for .GETMODE and .SETMODE system calls
;
MD_STATUS       defv    156     ; check whether input queues empty
MD_IFLUSH       defv    155     ; flush input queues
MD_FNKEYS       defv    152     ; turn function keys on or off
                                ;  d-register = 1 to enable fnkeys
                                ;  d-register = 0 to disable them
MD_PSIGHUP      defv    151     ; signal current process if hang up
;               defv    150     ; (this value reserved)
MD_MODEM        defv    148     ; (QTTYs and MTTYs only)
MD_TYP          defv    147     ; (TYP only)
```

```
; D-register values for MD_ISPEED baudrate calls
;
S_HANGUP        defv    0       ; hang up phone
;               defv    1       ; 50 baud
;               defv    2       ; 75 baud
S_110           defv    3       ; 110 baud
;               defv    4       ; 134.5 baud
S_150           defv    5       ; 150 baud
;               defv    6       ; 200 baud
S_300           defv    7       ; 300 baud
;               defv    8       ; 600 baud
S_1200          defv    9       ; 1200 baud
;               defv    10      ; 1800 baud
S_2400          defv    11      ; 2400 baud
S_4800          defv    12      ; 4800 baud
S_9600          defv    13      ; 9600 baud
;               defv    14      ; External A
;               defv    15      ; External B
S_19200         defv    16      ; 19200 baud
S_CTSWAIT       defv    125     ; wait for Clear To Send
S_NOCHG         defv    126     ; no change of baudrate
S_UNINIT        defv    127     ; baudrate has not been initialized yet
Sfl_AUTO        defv    7       ; (bit 7) input CRs from keyboard to set baudr

; D-register & e-register bits for MD_MODE1 calls
;
TANDEM          defv    0       ; send XOFF/XON to control filling of input buf
XTAB            defv    1       ; expand TABs
LCASE           defv    2       ; convert alphabetics to lower case
ECHO            defv    3       ; echo input
CRDEVICE        defv    4       ; on input, map CR into NL,
                                ; on output, change NL to CRLF.
RAW             defv    5       ; on input, return after each character,
                                ; no erase, linekill, or EOF characters,
                                ; no output PAUSE or output width truncation,
                                ; treat X-OFF & X-ON as regular input.
ODD             defv    6       ; parity function bits
EVEN            defv    7       ;

; D-register & e-register values for MD_MODED calls
;
NLDELAY         defv    03H     ; (pairs of bits)
TABDELAY        defv    0CH     ;
CRDELAY         defv    30H     ;
FFDELAY         defv    40H     ; (single bits)
BSDELAY         defv    80H     ;
```

```
; D-register & e-register bits for MD_MODE2 calls
;
PAUSE           defv    0       ; wait for CONTROL-Q after a page is output
NOTIMMECHO      defv    1       ; do not echo characters typed-ahead
NOECNL          defv    2       ; do not echo NLs
SGENABLE        defv    3       ; send SIGUSER signal if MD_USIGNAL key pushed
ABENABLE        defv    4       ; send SIGABORT signal if CONTROL-C key pushed
XFF             defv    5       ; expand FFs
WRAP            defv    6       ; wrap-around if page width is exceeded
SIGALLC         defv    7       ; send SIGUSER signal for every key pushed

; D-register & e-register bits for MD_MODE3 calls
;
ESCRETN         defv    0       ; ESC causes input line to be returned
FNKEYS          defv    1       ; response to 3102 function keys enabled
HUPENAB         defv    2       ; hang up modem when device is finally closed
SIGHUPALL       defv    3       ; send SIGHANGUP signals to all processes which
                                ; use this TTY device if modem hangs up
CBREAK          defv    4       ; on input, return after each character,
                                ; no erase, linekill, or EOF characters.
BINARY          defv    5       ; on input, return after each character,
                                ; no erase, linekill, or EOF characters,
                                ; no output PAUSE or output width truncation,
                                ; treat X-OFF & X-ON as regular input,
                                ; no tandem mode (i.e., no input buffer control),
                                ; no abort signal (CONTROL-C), no user signal,
                                ; no changing or checking parity bit,
                                ; no delays after control chars such as NLs,
                                ; no echoing,
                                ; no character transformations (i.e., ignore
                                ; the LCASE, CRDEV, and XTABS modes)
                                ; no function-key decoding.
CRIGNORE        defv    6       ; on output, ignore CR and change LF to CR
DISCARD         defv    7       ; discard the device when it is no longer open

; D-register bits for MD_STATUS calls
;
INOTEMPTY       defv    0       ; there is a character in the input buffer
                                ; (but if not CBREAK, RAW, or BINARY mode,
                                ; it won't be accessible until a whole line
                                ; is entered)

; .GETMODE d-register bits for MD_MODEM calls
;
RXDA            defv    0       ; Receiver Data Available
TXBE            defv    2       ; Transmitter Buffer Empty
DCD             defv    3       ; Data Carrier Detect
CTS             defv    5       ; Clear To Send
RXBREAK         defv    7       ; Receiver data line broken
```

```
; .GETMODE e-register bits for MD_MODEM calls
;
notRI           defv    6       ; Not ringing
notDSR          defv    7       ; Data Set not Ready

; .SETMODE d-register and e-register bits
;
RTS             defv    1       ; Request to Send
TXBREAK         defv    4       ; Break the transmitter line
DTR             defv    7       ; Data Terminal Ready

; D-register bits for MD_TYP call
;
TYPCHK          defv    1       ; the 3355 printer is in a check condition
TYPPAP          defv    2       ; the 3355 printer is out of paper
TYPRIB          defv    3       ; the 3355 printer is out of ribbon
TYPOFL          defv    4       ; the 3355 printer is off-line

; D_register values for MD_IDENT calls
;
ID_TTY          defv    0       ; Tuart terminal
ID_QTTY         defv    1       ; Quadart or Octart terminal
ID_LPT          defv    2       ; Parallel printer
ID_TYP          defv    3       ; Fully formed printer
ID_SLPT         defv    4       ; Serial printer
ID_QSLPT        defv    5       ; Serial printer on quadart
ID_CNET         defv    6       ; CNET driver
ID_FFP          defv    7       ; FFP processor driver
ID_SYSTEM       defv    8       ; System device
ID_TIMER        defv    9       ; Timer device
ID_TAPE         defv    10      ; Half-inch tape drive
ID_SCC          dev     11      ; SCC terminal
                                ; Values 12 through 127 reserved
                                ; Values 128 through 255 reserved for user-
                                ; defined drivers and devices

        list    xref    ; (use this line only with ASMB version 3.08 or later)
        list    on
```

/EQU/BMODEEQU.Z80

```
        list    off, noxref

;
; Cromemco Inc.
; August 24, 1985
;
; Mode definitions for block devices
;

; C-register values for .GETMODE and .SETMODE system calls
;
BMD_STATUS      defv    0       ; Get/set status byte
BMD_FLG1        defv    1       ; Get/set flag1 byte
BMD_FLG2        defv    2       ; Get/set flag2 byte
BMD_FLG3        defv    3       ; Get/set flag3 byte
BMD_SIZE        defv    4       ; Get number of bytes on device
BMD_SEEK        defv    5       ; Seek
BMD_INIT        defv    6       ; Initialize track
BMD_PRDWRT      defv    7       ; Primitive read/write
BMD_RDWRT       defv    8       ; Special read/write
BMD_SEEK        defv    8       ; Seek
BMD_RPM         defv    9       ; Get RPM
BMD_VERSION     defv    10      ; Version number
BMD_PHYCHAR     defv    11      ; Physical characteristics
BMD_LDFIRM      defv    12      ; Load firmware
BMD_SOFT        defv    13      ; Accumulated number of retries
BMD_HARD        defv    14      ; Accumulated number of hard errors
                                ; Values 15 through 127 reserved
                                ; Values 128 through 255 reserved for user-
                                ; supplied drivers
BMD_RETRY       defv    15      ; Number of retries before hard error
                                ;
                                ; Values 17 .. 63 reserved
                                ; Values 64 .. 127 special device modes
                                ; Values 128 .. 255 reserved for user
                                ; supplied drivers

; Floppy tape special number
;

BMD_RETEN       defv    64      ; Number of tape repositions before
                                ;       a retension

; IMI disk special numbers
;

BMD_IMITYPE     defv    64      ; Get type of IMI drive

; Values returned by BMD_IMITYPE
;
```

```
IM_50070        defv    01H     ; IMI model 50070
IM_5007W        defv    02H     ; IMI model 5007W
IM_5018H        defv    03H     ; IMI model 5018H
IM_7710A        defv    04H     ; IMI model 7710A
IM_7710B        defv    05H     ; IMI model 7710B

; D-register & e-register bits for BMD_STATUS calls
;
DS.BUSY         defv    0       ; Device Busy (in use)
DS.WANT         defv    1       ; Device Wanted (do wakeup)
DS.READ         defv    2       ; Read-only device
DS.MODF         defv    3       ; Super-block modified
DS.MOUNT        defv    4       ; Device mounted
DS.HOME         defv    5       ; Device has been homed
DS.BFSTEP       defv    6       ; Buffered step flag
DS.VERIFY       defv    7       ; Verify after write

; D-register & e-register bits for BMD_FLG1 calls
;
DF.SMALL        defv    0       ; 1=small floppy 0=large floppy
DF.DSIDE        defv    1       ; Double sided
DF.DDENS        defv    2       ; Double density
DF.DTRACK       defv    3       ; Double tracked
DF.CROMIX       defv    4       ; Cromix format disk
DF.CDOS         defv    5       ; Cdos format disk
DF.BACKUP       defv    6       ; Backup format disk
DF.VOICE        defv    7       ; 0=step 1=voice coil

; D-register values for BMD_FLG2 calls
;
D2.SMALL        defv    0       ; Small floppy
D2.LARGE        defv    1       ; Large floppy
D2.STDC         defv    2       ; STDC hard disk
D2.FSMD         defv    3       ; Fixed part of SMD hard disk
D2.RSMD         defv    4       ; Removable part of SMD hard disk
D2.UNIFORM      defv    5       ; Uniform floppy
D2.MEMORY       defv    6       ; Processor memory
D2.RAM          defv    7       ; RAM disk
D2.FTAPE        defv    8       ; Floppy tape
D2.IMI          defv    9       ; IMI hard disk

; D-register & e-register bits for BMD_FLG3 calls
;
D3.WRTPRO       defv    0       ; Device is write protected
D3.INTRPT       defv    1       ; Device interrupts
D3.DUAL         defv    2       ; Dual drive

;  Floppy minor device number bits
;
FDENSITY        defv    6       ; 0 = double density
FSIDES          defv    5       ; 0 = double sided
FDUAL           defv    4       ; 1 = dual drive (PERSCI)
FDTRACK         defv    3       ; 1 = double tracked
```

```
FSIZE           defv    2       ; 0 = 8", 1 = 5"
FUNIT           defv    03H     ; Mask for unit number

;  SMD minor device number bits
;
CONTROLLER      defv    7       ; Controller number
DRIVE           defv    6       ; Drive number
FIXED           defv    5       ; Fixed flag
PARTITION       defv    1FH     ; Partition number mask

; Data structure for BMD_INIT call
;
        struct  0
in.flags        defs    2       ; Density from minor device number
in.side         defs    2       ; Side to initialize
in.track        defs    2       ; Track to initialize
in.buf          defs    4       ; Pointer to track image
in.size         defs    0       ; Size of structure
        mend

; Data structure for BMD_SEEK call
;
        struct  0
sk.status       defs    1       ; Controller status
sk.ferror       defs    1       ; Fatal error number
sk.serror       defs    1       ; System error number
sk.verify       defs    1       ; Verify seek flag
sk.side         defs    2       ; Side
sk.track        defs    2       ; Track
sk.size         defs    0       ; Size of structure
        mend

; Data structure for BMD_RDWRT call
;
        struct 0
rw.read         defs    2       ; Read/write flag
rw.buf          defs    4       ; Buffer pointer
rw.number       defs    4       ; Number of blocks
rw.blknr        defs    4       ; Starting block number
rw.size         defs    0       ; Size of structure
        mend

; Data structure for BMD_PRDWRT call
;
        struct  0
prw.status      defs    1       ; Controller status
prw.ferror      defs    1       ; Fatal error number
prw.serror      defs    1       ; System error number
prw.read        defs    1       ; Read/write flag
prw.buf         defs    4       ; Buffer pointer
prw.number      defs    2       ; Number of sectors to do
prw.sector      defs    2       ; Starting sector number
prw.surface     defs    2       ; Surface number to read/write
prw.cylinder    defs    2       ; Cylinder number to read/write
prw.size        defs    0       ; Size of structure
        mend

; Status bits for primitive operations
;

sts.ioerror     defv    0       ; IO error
sts.nioerror    defv    1       ; Not IO error
sts.select      defv    2       ; Error on select
sts.seek        defv    3       ; Error on seek
sts.prd         defv    4       ; Error on primitive read
sts.pwr         defv    5       ; Error on primitive write
sts.ptx         defv    6       ; Error on transfer

;
; Floppy status bits
;
fls.select      defv    1       ; Error on select
fls.home        defv    2       ; Error on home
fls.rdadd       defv    3       ; Error on read address
fls.seek        defv    4       ; Error on seek
fls.pread       defv    5       ; Error on preread
fls.read        defv    6       ; Error on read
fls.write       defv    7       ; Error on write
fls.wtrk        defv    8       ; Error on write track
;
; SMD status bits
;
sms.select      defv    1       ; Error on select
sms.home        defv    2       ; Error on home
sms.seek        defv    3       ; Error on seek
sms.read        defv    4       ; Error on read
sms.write       defv    5       ; Error on write
sms.head        defv    6       ; Error on select head
sms.pread       defv    7       ; Error on preread

; Data structure for BMD_PHYCHAR call
;
        struct  0
phy.surface     defs    2       ; Number of surfaces on device
phy.cylinder    defs    2       ; Number of cylinders on device
phy.sector      defs    2       ; Number of sectors/track
phy.secsiz      defs    2       ; Number of bytes/sector
phy.size        defs    0       ; Size of structure
        mend

; Data structure for BMD_LDFIRM call
;
        struct  0
ldf.flags       defs    2       ; Flags (see below)
ldf.count       defs    2       ; Number of bytes
ldf.buf         defs    4       ; Pointer to firmware
ldf.size        defs    0       ; Size of structure
        mend

LDFRM_DEBUG     defv    8DH     ; Load debugger firmware
LDFRM_FIRM      defv    8FH     ; Load regular firmware

        list xref, on

;       Tmodeequ.h:     Cromemco 68000 C I/O header file
;       Copyright (c) 1984 by Cromemco, Inc., All Rights Reserved
;
;       This file contains declarations of all values which are
;       used in the getmode and setmode Cromix system calls, for
;       TP tape devices.
;
;       Dec-18-84
;

TPABORT         defv    196     ; re-initialize tape driver
TPFMARK         defv    198     ; write file mark
TPSECURE        defv    199     ; security erase
TPREWIND        defv    200     ; rewind
TPUNLOAD        defv    201     ; rewind and unload
TPMODE          defv    202     ; mode bits
TPFILNO         defv    203     ; file number
TPBLKNO         defv    204     ; block number
TPOBLKLN        defv    205     ; block length for next block written
TPIBLKLN        defv    206     ; block length of first block read
TPOBLKS         defv    207     ; number of blocks written
TPSTAT          defv    208     ; get error (status-2, status-1)

;       TPMODE bits

EOFCLOSE        defv    7       ; write EOF to tape when device closes

;       TPSTAT status bits  (obtained from PIO input port A)
;       These bits are returned in e-register
;       Old names are without leading TP

TPDRVBUSY       defv    7       ; drive busy
TPWRRDY         defv    6       ; FIFO ready for input (used for write)
TPRDRDY         defv    5       ; FIFO output ready (used for read)
TPLOADPT        defv    4       ; load point
TPFBUSY         defv    3       ; formatter busy
TPONLINE        defv    2       ; on line
TPIDENT         defv    1       ; ident
TPRDY           defv    0       ; ready

;       TPSTAT status bits  (obtained from PIO input port B)
;       These bits are returned in e-register
;       Old names are without leading TP

TPHISPEED       defv    7       ; high speed status
```

```
TPHARDERR      defv    5        ; hard error
TPFLMARK       defv    4        ; file mark
TPCORERR       defv    3        ; correctable error
TPWRPROT       defv    2        ; file write-protected
TPEOT          defv    1        ; end of tape
TPRWINDING     defv    0        ; rewinding
```

Appendix C

ASCII CHARACTER CODES

HEX	CHARACTER	HEX	CHAR	HEX	CHAR	HEX	CHAR
00h	NUL (CONTROL-@)	20h	SPACE	40h	@	60h	'
01h	SOH (CONTROL-A)	21h	!	41h	A	61h	a
02h	STX (CONTROL-B)	22h	"	42h	B	62h	b
03h	ETX (CONTROL-C)	23h	#	43h	C	63h	c
04h	EOT (CONTROL-D)	24h	$	44h	D	64h	d
05h	ENQ (CONTROL-E)	25h	%	45h	E	65h	e
06h	ACK (CONTROL-F)	26h	&	46h	F	66h	f
07h	BEL (CONTROL-G)	27h	'	47h	G	67h	g
08h	BS (CONTROL-H)	28h	(	48h	H	68h	h
09h	HT (CONTROL-I)	29h	)	49h	I	69h	i
0Ah	LF (CONTROL-J)	2Ah	*	4Ah	J	6Ah	j
0Bh	VT (CONTROL-K)	2Bh	+	4Bh	K	6Bh	k
0Ch	FF (CONTROL-L)	2Ch	,	4Ch	L	6Ch	l
0Dh	CR (CONTROL-M)	2Dh	-	4Dh	M	6Dh	m
0Eh	SO (CONTROL-N)	2Eh	.	4Eh	N	6Eh	n
0Fh	SI (CONTROL-O)	2Fh	/	4Fh	O	6Fh	o
10h	DLE (CONTROL-P)	30h	0	50h	P	70h	p
11h	DC1 (CONTROL-Q)	31h	1	51h	Q	71h	q
12h	DC2 (CONTROL-R)	32h	2	52h	R	72h	r
13h	DC3 (CONTROL-S)	33h	3	53h	S	73h	s
14h	DC4 (CONTROL-T)	34h	4	54h	T	74h	t
15h	NAK (CONTROL-U)	35h	5	55h	U	75h	u
16h	SYN (CONTROL-V)	36h	6	56h	V	76h	v
17h	ETB (CONTROL-W)	37h	7	57h	W	77h	w
18h	CAN (CONTROL-X)	38h	8	58h	X	78h	x
19h	EM (CONTROL-Y)	39h	9	59h	Y	79h	y
1Ah	SUB (CONTROL-Z)	3Ah	:	5Ah	Z	7Ah	z
1Bh	ESC (CONTROL-[)	3Bh	;	5Bh	[	7Bh	{
1Ch	FS (CONTROL-\)	3Ch	<	5Ch	\	7Ch	|
1Dh	GS (CONTROL-])	3Dh	=	5Dh	]	7Dh	}
1Eh	RS (CONTROL-^)	3Eh	>	5Eh	^	7Eh	~
1Fh	US (CONTROL-_)	3Fh	?	5Fh	_	7Fh	DEL

NUL	= null	DC1	= device control 1
SOH	= start of heading	DC2	= device control 2
STX	= start of text	DC3	= device control 3
ETX	= end of text	DC4	= device control 4
EOT	= end of transmission	NAK	= negative acknowledge
ENQ	= enquiry	SYN	= synchronous idle
ACK	= acknowledge	ETB	= end transmission block
BEL	= bell	CAN	= cancel
BS	= backspace	EM	= end of medium
HT	= horizontal tab	SUB	= substitute
LF	= line feed	ESC	= escape
VT	= vertical tab	FS	= file separator
FF	= form feed	GS	= group separator
CR	= carriage return	RS	= record separator
SO	= shift out	US	= unit separator
SI	= shift in	SP	= space
DLE	= data link escape	DEL	= delete

Reader Responses To This Documentation

Dear Reader,

We have made a sincere effort to provide you with the information you need in this manual. If you should find the documentation deficient or in error, let us know so we can correct it. We appreciate and value your response; it will be useful in improving the documentation. Please detach and use the Reader Response Card below to send us your comments.

Thank you for your time and interest in Cromemco products.

Sincerely,
Winthrop A. Stiles III
Technical Publications Manager

(Detach Here)

Cromemco®

Reader Response Card

To: Winthrop A. Stiles III,
Technical Publications Manager

Re (Manual title):____

My System is (Specify configuration):______________________________

The following information is incorrect (Please specify page number):__________

____

____

____

____

____

____

(Detach Here)

(Fold Here)

The following additional information would be helpful:____________________

____

____

What general suggestions do you have for improving this manual?______________

____

____

If you need a response from Cromemco, please print your name, mailing address, and telephone number:

Name:______________________________

Address:____________________________

____

____

Telephone: (____)__________________________

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 599 MOUNTAIN VIEW, CA

POSTAGE WILL BE PAID BY ADDRESSEE

Cromemco®

Attn: Winthrop A. Stiles III
Technical Publications Manager
280 Bernardo Avenue
P.O. Box 7400
Mountain View, CA 94039

(Fold as indicated and tape the edge)

Cromemco®

280 Bernardo Ave.
P.O. Box 7400
Mountain View, CA 94039