NISTIR 89-4023

Small Computer System Interface (SCSI) Command System

Software Support for Control of Small Computer System Interface Devices

Documentation: John Gorczyca

Software: John Gorczyca and Eduardo Sanchez Villagran

U.S. DEPARTMENT OF COMMERCE
National Institute of Standards and Technology
(Formerly National Bureau of Standards)
National Computer Systems Laboratory
Advanced Systems Division
Gaithersburg, MD 20899

January 1989

NISTIR 89-4023

Small Computer System Interface (SCSI) Command System

Software Support for Control of Small Computer System Interface Devices

Documentation: John Gorczyca

Software: John Gorczyca and Eduardo Sanchez Villagran

U.S. DEPARTMENT OF COMMERCE
National Institute of Standards and Technology
(Formerly National Bureau of Standards)
National Computer Systems Laboratory
Advanced Systems Division
Gaithersburg, MD 20899

January 1989

National Bureau of Standards became the National Institute of Standards and Technology on August 23, 1988, when the Omnibus Trade and Competitiveness Act was signed. NIST retains all NBS functions. Its new programs will encourage improved use of technology by U.S. industry.

U.S. DEPARTMENT OF COMMERCE
C. William Verity, Secretary

NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
Ernest Ambler, Director

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

I. F O R E W O R D:

The National Computer and Telecommunications Laboratory of the National Institute of Standards and Technology (NCTL/NIST) is currently assisting the National Archives and Records Administration in developing a testing methodology that can be used to predict life expectancy of optical disk media. This testing methodology can be applied in evaluating one or more physically different types of optical digital disks. The results of such testing will assist government managers in planning how long information may be stored on optical disk media without significant degradation.

To develop this testing process, it is first necessary to develop methods for the examination of media characteristics. These tests will determine if the media characteristics are changing after submitting the media to different aging and stress simulations, such as heat, humidity and pressure. Further investigation into media defect mechanisms will also be necessary.

Tests for media characteristics such as the bit/byte error rate and the carrier-to-noise ratio are being measured at this time using commercially available drives. Two different types of media are currently being studied, one disk is a poly-carbonate substrate based media and the other is based on a glass substrate. Other types of media, such as magneto optic media, will be incorporated into the program at a later date. The media made of the poly-carbonate substrate is SONY 12" WORM (Write Once Read Many) media and runs in a SONY WDD-3000 disk drive. The controller for this SONY drive is the WDC-2000-10/A. The glass media being researched is the ATG Gigadisc, also a 12" WORM disk, that runs in ATG GC/GD 1001, the drive/controller unit. Both drive controllers communicate to their hosts through a Small Computer System Interface (SCSI).

SCSI control software with a large number of commands and convenient data access would clearly aid this type of study. This publication describes the SCSI Command System that provides variable structures and a library of SCSI commands for software assisted device control. It was designed and developed by NCTL/NIST personnel. Written on the IBM PC/AT, the software utilizes a Micro Design International SCSI HA-1 PC host adapter to establish bus communications. The SCSI Command System enables the control of the SCSI bus and input/output data processes.

II. D I S C L A I M E R

Because of the nature of this report, it is necessary to mention vendors and commercial products. The presence or absence of a particular trade name product does not imply criticism or endorsement by the National Institute of Standards and Technology, nor does it imply that the products identified are necessarily the best available for the purpose.

SYSTEM DEVELOPMENT

1 SYSTEM DEVELOPMENT

1.1 PROBLEM:

The Institute's goals demanded that a software assisted SCSI device control system be created. The software for this control system should have the ability to execute any SCSI command or any manufacturer unique command for the SCSI bus. It should also have the ability of convenient and precise data access. Data should be available so that it can be analyzed on a byte by byte basis. System programming would be most beneficial if it could be done in the 'C' language. Also, system programming should allow the ability to create and utilize libraries of functions. These features would ideally be present in an economical, portable, and practical system.

1.2 SOLUTION:

Working with the Micro Design International (M.D.I.), HA-1, SCSI host adapter, NCTL/NIST personnel first developed a function to send commands on the SCSI bus. The function was written in assembly language and calls the ROM entry point of the M.D.I. host adapter. The call to ROM sends a pointer to a command block structure as defined in the M.D.I. host adapter manual. The card's ROM then drives the command information onto the SCSI bus where it can be received by the appropriate devices. The function was named 'cif', for the 'Command Interfacing Function'. This routine can be accessed from 'C' language.

After successfully writing this function, it was necessary to develop the variable structures to be used within the system. The structures must be accessible globally for use by 'cif'. 'C' language programming allowed easy variable passing and simplest command/data interfacing. Two system structures were created; the command block named 'cmdblk' and the buffer called 'buffer'. The buffer structure can be explained easily. There are 10K bytes which the system relies on for all data transactions. The command block structure is more sophisticated. Its members include bytes reserved for Micro Design International's driver codes. Additionally, environment parameters such as the buffer's length, offset and segment in memory are configured here. The command block structure directs the command to the proper device, using its information regarding target and logical unit number. This structure is modeled after the form that is presented in the Micro Design International HA-1 host adapter manual.

The next step in system development was to utilize the command interfacing function and system structures, to establish a library of commands. As preliminary steps to this aspect of system construction, support files and functions to set up the system's environment were generated.

Two files were created for the purpose of defining system structures and functions. The files are to be included in programming for the system. The file "scsi.h" contains the parameter syntax of the command library and enables proper calling and value passing. The file called "def.h" is used to define the command block and buffer structures. It enables the usage of library functions and allows the usage of system variables in programming.

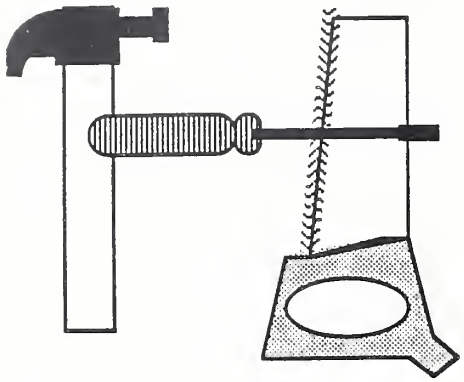
Subroutines were created to properly set command block structure members. These library utilities enable user interactions to configure the system and edit the system's operation parameters. A function called 'tidlu' was created to assign the target id and the logical unit number. Other functions assume the target and logical unit numbers to be set before they are executed. 'Setbufd' was developed next. This function, through its proper configuration of the buffer parameters in the command block structure, establishes a buffer which the system uses for handling all input and output of data. Returning error codes were dealt with by having most library functions return a flag indicating the command's success or failure. A function called 'erdec' was designed to retrieve and decode error data. It supplies a message regarding exactly what failure occurred. By eliminating the need to reference a technical manual and/or examine the command block, 'erdec' saves time and simplifies the debugging of programs for the system.

After development of these preliminary system files and functions was completed, a library of functions was built. This was done with the aid of Microsoft C's and Turbo C's library management software. Because of hardware considerations complete sets of commands for SONY and ATG drives were included. In addition, many commands from the SCSI standard were included. The commands provided form a well rounded set that can accomplish many tasks. For the sake of convenience data manipulation functions which store, display and recall data were also developed.

The list of functions contained in section two of the user's guide, shows exactly what was included in the writing of the library. The list presents the function's name, a brief description and the function type. A function type can be SONY, ATG, SCSI, or TOOL [see figure 1]. These classifications arose naturally from the origin and/or run time result of each library member.

After the library was created, the system evolved into what could be called two levels of usage; the level that can create libraries, and the level that uses them. The upper level entails the usage of the library functions. Lower level programming implies interactions of the system's variable structures (figures 2 and 3 illustrate this hierarchy). The entire library is based

LIBRARY FUNCTION TYPES

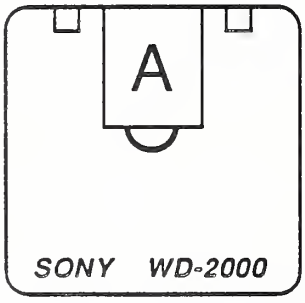


1 **TOOLS**

-DO NOT COMMUNICATE WITH DEVICES
-DIRECT, MODIFY, OR SUPPLY DATA

2 **ATG**

-MANUFACTURER'S COMMANDS



3 **SONY**

-MANUFACTURER'S COMMANDS

4 **SCSI**

- COMMANDS FROM THE SCSI STANDARD FOR ALL SCSI DEVICES.

FIGURE 1

on this style of programming. The two levels of programming are different, but are in no way isolated from one another. Most programs contain both programming styles, mixing the levels, and allowing them to interact. Figure 4 illustrates the system programming possibilities.

An example of the lower level programming style is a function created in the writing of the library. Below is an actual section of the library's source code, it is the short read function:

```
/*****************************************************/
/* S H O R T   R E A D     F U N C T I O N           */
/*****************************************************/

sread(len,addr)
long addr;                  /* receives address          */
byte len;                   /* receives the length       */

/* function assumes a buffer is set up                   */
/* function assumes that the target/lun are set          */
/* function send the short read command                  */

{
byte ah,am,al;
clear();                    /* clear the command block   */
ah=mh(addr);                /* high byte of addr.        */
am=ml(addr);                /* middle byte of addr.      */
al=lo(addr);                /* low byte of addr.         */

/* T H E   S H O R T   R E A D   C O M M A N D           */
cmdblk.cdb[0]=0x08;
cmdblk.cdb[1]=(cmdblk.cdb[1] & 0xe0) | ah;
cmdblk.cdb[2]=am;
cmdblk.cdb[3]=al;
cmdblk.cdb[4]=len;
/*****************************************************/
return(cif(&cmdblk));     /* sends the command         */
}
```

This function is sent the length of the read to occur and the address at which to read. The long unsigned integer representing the address is broken into its separate bytes using library functions. The command block is set up and a call to the interfacing function is made to send this data. Note that this section of code contains no "include" type files. This is because all definitions necessary for its use occur elsewhere in the same module of code. The remainder of the function library was developed with lower level programming similar in structure to the example above.

SYSTEM ORIENTATION

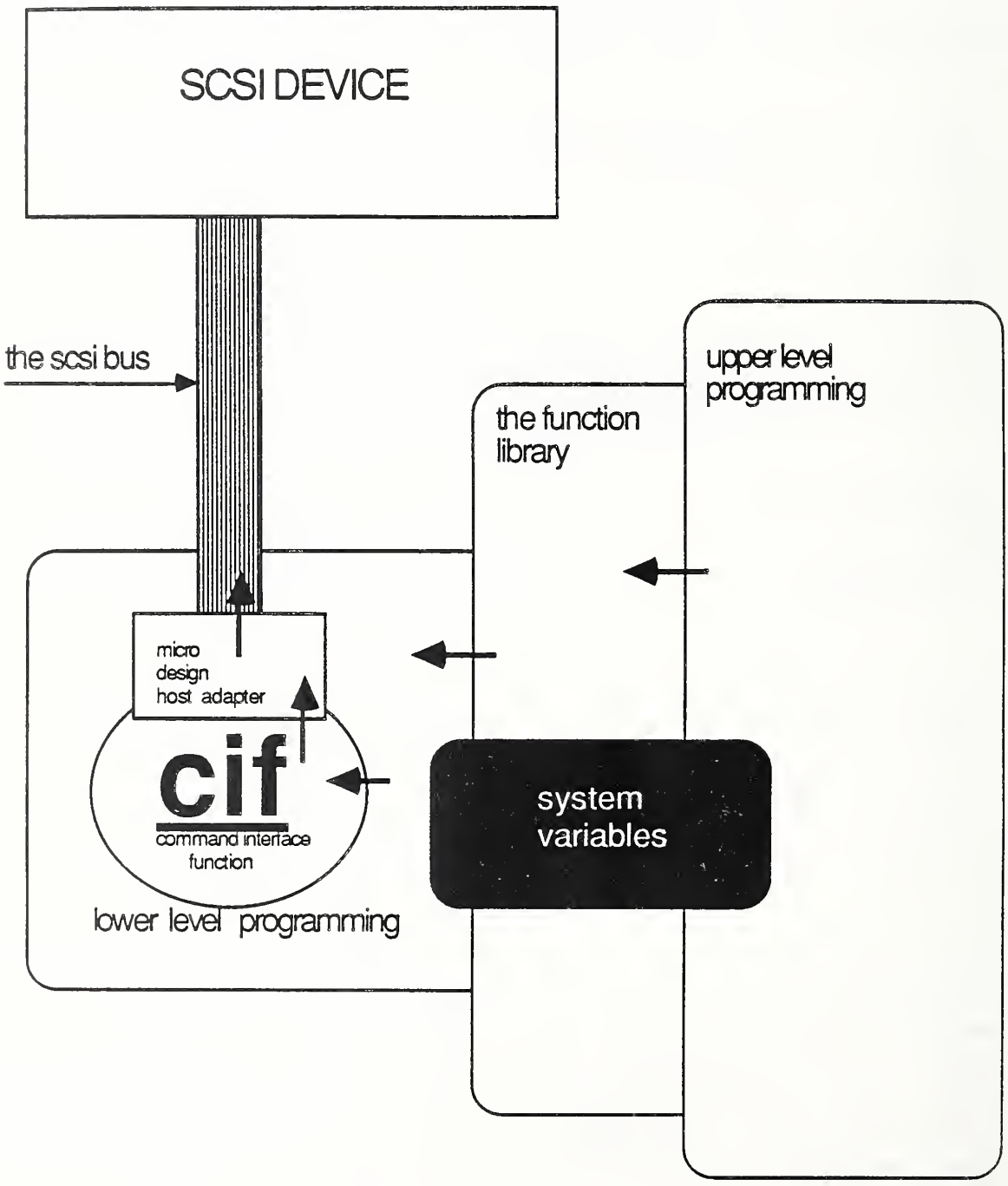


FIGURE 2

SYSTEM PROGRAMMING

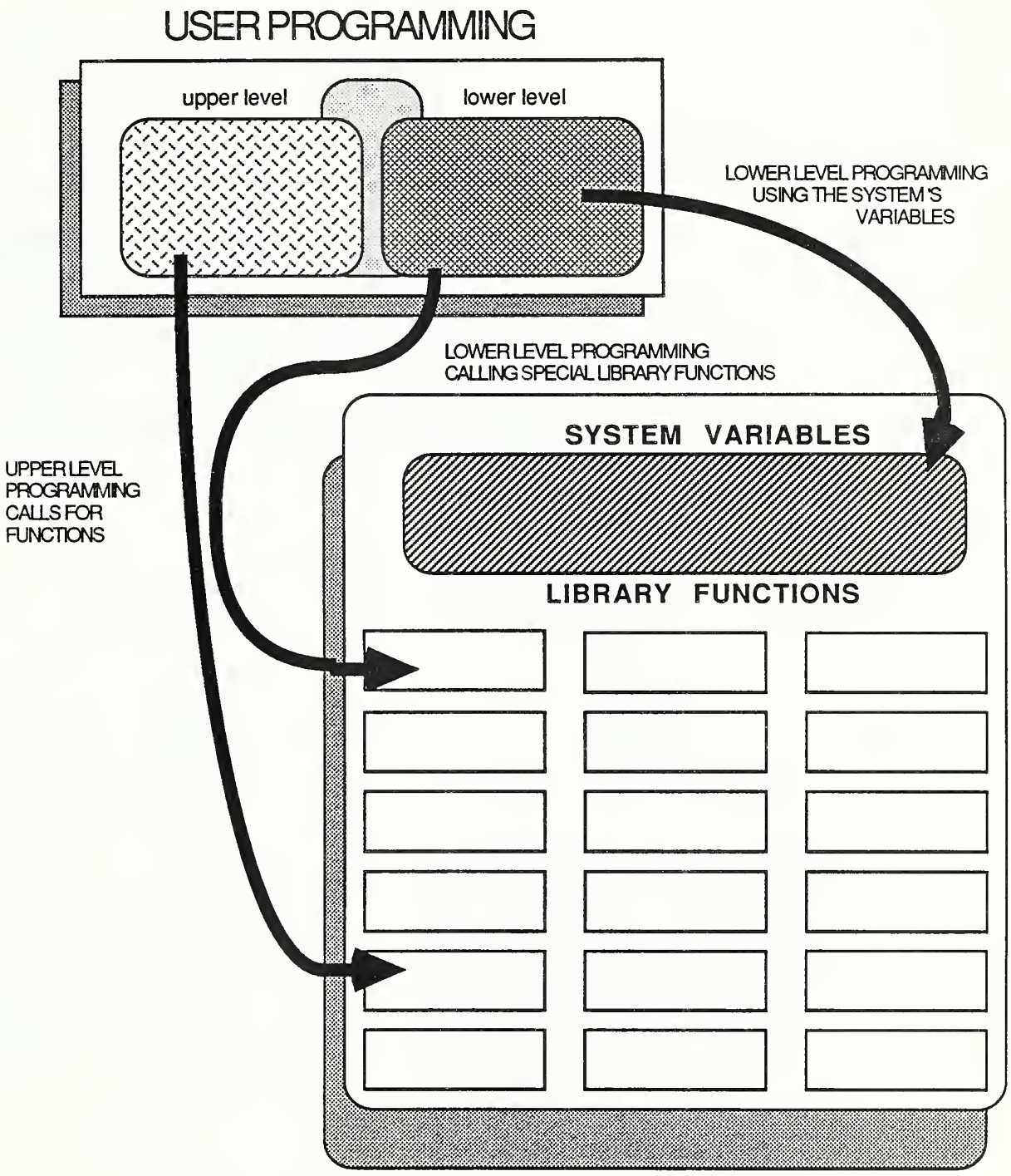
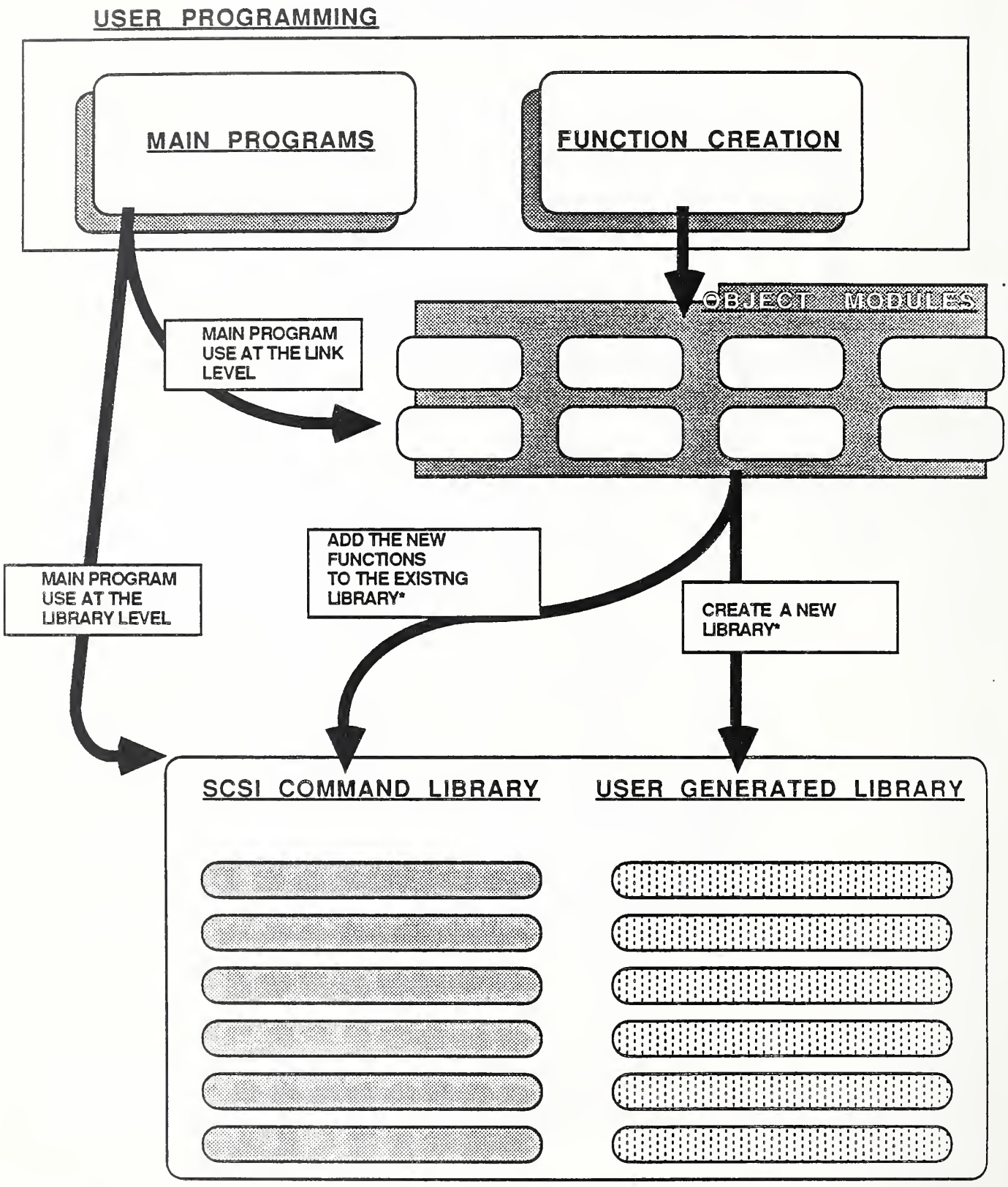


FIGURE 3

SYSTEM USAGE AND OVERVIEW



* OPTION POSSIBLE ONLY WITH THE USE OF A LIBRARY MANAGER PROGRAM

FIGURE 4

Upper level programming deals with the library of functions. An example in this style of programming was developed in the early stages of system use:

```
/*********************************************************/
/* example program #1                                    */
/* the classic read example                              */
/*                                                       */

#include "scsi.h"          /* must be included           */
#include "def.h"           /* must be included           */

main()
{

setbufd(1024);       /* establish a 1024 byte buffer     */
tidlu(0,0);          /* target=0 logical unit=0          */
sarsto(1);           /* start the drive                  */
sread(1,0x203L);     /* read logical sector 203hex, 1K   */
sarsto(0);           /* stop the drive                   */
displbu();           /* display the buffer on the screen*/
makebuf("b:data.dat"); /* make a disk copy of the buf    */
prinbuf();           /* print a buffer hard copy         */

}                    /* end of program                   */
/*                                                       */
/*********************************************************/
```

This example gives a basic idea of how the functions of the library can be used. Many programs similar to this were developed in the testing that took place to verify each library command.

Upon completion of the software, a manual describing the system and its proper use was written. This user's guide is intended to provide information regarding proper library use and function syntax. Another portion of text discusses lower level programming and system variable uses. In depth examples of both programming levels are provided. In addition, the function library section describes each function and provides information allowing the user to interact with the system more quickly and easily. Source code and installation instructions (for Turbo and Microsoft C) are included with this user's guide.

1.3 A P P L I C A T I O N S:

Currently the system is being used for many differing applications. The system is being run in both TURBO C and Microsoft 'C' based machines. Programming for byte/bit error rate has been developed for SONY and ATG hardware using the library of commands and the system's variables in the Microsoft 'C' machine. Diagnostic test programs for this hardware have also been written using the system, enabling examination of the hardware's state, signals and current operational parameters. Individual users are making their own libraries based on the system to meet their specific needs. In addition, the original command library is under constant expansion as new command ideas develop. The system will have many other useful applications with respects to SCSI hardware and data analysis in the future.

USER'S GUIDE

2 USER'S GUIDE

2.1 SYSTEM OVERVIEW:

The SCSI Command System has many useful features that are a direct result of the Institute's goals. The system is very portable. It is compact and yet remains very flexible, providing the possibility of implementing any SCSI command, be it standard or manufacturer unique. The system is based on programming that is close to the data describing these commands. By providing simple user access to one byte or an entire buffer, the software also furnishes powerful input/output data capabilities. Data examination tools and a large set of commands are provided in the function library. Others can easily be created. The structure and focus of the included library is intended to provide power with easy programming. Because commands are specific in nature, it takes a minimal amount of programming to implement them. The command function library offers many convenient and practical programming benefits. As for the SCSI Command System, it offers unlimited power in a sense that any command can be implemented, and all data can be accessed with ease.

Programming with the system will be discussed as two levels of interaction: lower level and upper level. Both levels utilize the command interface function named 'cif'. This function is a library member and is utilized by most of the remaining library functions. The command interface routine makes device control with this system possible. Note that the two levels of programming are different, but are in no way isolated from one another. A program for the system can, and usually does, contain both levels of programming, referencing low level system variables and calling members of the function library.

The term 'upper level' programming implies interaction with the system's library of functions. The library was written in the lower level style. It sets up commands and calls the interfacing function 'cif'. Parameters passed to the routines enable proper execution. For example, a function may require an address, a length or both. Upper level programming techniques provide the user with a powerful yet convenient method for developing programs. It is, however, somewhat more isolated from data, providing a limited set of data display functions. Standardized functions to make and recall data files on floppy disk are library members and are very useful. There are also 'include' type files for use in programs constructed at this level. The library offers many benefits in the form of preprogrammed conveniences and is constructed with lower level programming.

Lower level programming deals directly with the system variables representing the input/output data and the command to

be sent to an external SCSI device. Command execution occurs after command variables have been properly configured. 'Cif' can then be called to establish the command interface, that is, send the command. In this way, new functions can be created for the library; useful object modules can be established as well. The lower level of programming is very close to the input/output data, enabling easy displays and examinations of data. Input/output data can also be used as inner program parameters, limiting a loop or directing a program's path for instance. There are 'include' type files and functions provided in the library that further simplify this level of programming.

2.2 USER'S GUIDE OVERVIEW:

This user's guide is divided into two sections. The first discusses system programming. After important preliminary steps for all programing are examined, upper level programming is discussed. Offering insight into buffer usage for input and output, this section provides an understanding necessary for programming in the upper level style. Additionally, it offers information helpful for the lower level programming style. Essential programming steps are presented and discussed. Examples are provided and fully examined.

Lower level programming is the next topic discussed in section one. Its direct interaction with command and input/output data, and the very important command interfacing function are explored. The variable structures of the library that are constantly dealt with in this programming style are examined and their members are clearly defined. Library functions essential to low level programming and 'include' file usage are discussed to establish proper programming techniques for this aspect of the system. Examples directly from the included library's source code, as well as other examples are provided.

The second section serves as a reference for the function library. It contains all available functions, defining their purpose and parameters in a standard page format. Additional sections provide usage hints and examples of 'C' language calls. A listing of each function's syntax is also presented for reference purposes.

USER'S GUIDE

section one

2.3 PROGRAMMING OVERVIEW

When programing with this library there are programing steps which must be included in all programs. This overview section presents information that will be useful to both levels of programming. It is presented as a preliminary to all programming to be done with the system.

2.3.1 COMMAND INTERFACING FUNCTION

The Command Interfacing Function is the heart of the SCSI Command System. It interfaces the variables of the system to the ROM present on the host card, which in turn drives the command. The library functions set up the command structure and call this function. Low level programming will also utilize 'cif' to implement non-library commands for SCSI devices. 'Cif' is presented in the function library portion of text. Its description states that it must always send a pointer to a command. The pointer reveals any command that the command block structure currently contains. The format of this command block is presented in the lower level programming description and is also described in the Micro Design International host adapter manual.

2.3.2 INCLUDE FILES AND STRUCTURES

"Scsi.h" and "def.h" are two files that are essential to proper program execution. They must be included in system programs for compilation and linkage to take place. "Scsi.h" contains the syntax definitions of all the functions in the library. "Def.h" is a structural definition block that allows the buffer and command block structures (the system's variables) to be accessed directly in main programs. The benefits of direct library variable access are discussed in the lower level programing portion of the manual.

All programs or functions intended for use with the library of functions must include the two ".h" header files mentioned above. They define and make accessible the structures that are used throughout the programing, enable all library functions, and allow proper compilation and linkage. They should be included outside any main program intended for the system as follows;

```
#include "scsi.h"
#include "def.h"
main()
{
```

or outside a function that uses library functions such as:

```
#include "scsi.h"
#include "def.h"
name()
{
```

The above programing statement will be used most frequently and is valid for programs and functions that use the system's library and variables. Once these files have been included, proper compilation will occur. Linkage to the library will resolve unknown externals when these files have properly been included.

In lower level cases where a function is created that uses no library members, the following statement can be included:

```
#include "def.h"
name()
{
```

This allows system variable usage within the function only. No library functions are anticipated with this type of 'include' declaration.

The examples provided can be examined to yield a further understanding of include file usage at both the upper and lower levels.

2.3.3 BUFFER USAGE AND I/O

The program established buffer is used in all aspects of input and output. The buffer must always be established before any data transfer can occur. Not only read and writes require the buffer, many of the library functions use it for a parameter list or an allocated memory location for returning data. It will frequently be used in lower level programs and functions as well. Input of new data to the buffer will supercede any information currently stored there.

A read command needs to have a buffer established so that data read from a device will have some place to be stored. Data retrieved during a read will be placed in the buffer. The write command writes the data from the buffer to the medium. If there is no buffer there is no data to write. If the buffer has not been loaded, there is still no data to write. Unloaded, the buffer is filled with random values(usually zero). Once the buffer is loaded, a write command will transfer the desired data. Data is written from the buffer to the medium.

A request sense command returns sense data. In the case of this system, it does so into the data buffer. Likewise any command that returns values, e.g, mode sense, first blank sector search, receive diagnostic results, etc., will do so in the buffer. Like any buffer data, device parameter data can be examined with the buffer output commands.

Consider a mode select command. A mode selection requires a parameter list describing the mode to be selected. The data for this selection must be placed in the program buffer before this function is called. Likewise any other commands that require a data list will look for the data in the buffer. Send diagnostics, a SONY command, also works as above. The subcommands for this command must be present in the buffer before execution occurs.

The simple upper level examples provided show how a buffer works in all the scenarios mentioned. With a limited amount of use it becomes apparent that the buffer is used for all data at both levels of system use. If a function requires a list of data, it will look for it in the buffer. If it returns any kind of data, it will be returned there.

2.3.4 PROGRAM COMPILING AND LINKING

Main programs should contain the "include" type files for proper compilation and linkage. System use was confirmed with Microsoft C and TURBO C. The following compilation/linkage instructions for system programs are dedicated to these two 'C' language systems.

TURBO C users should create programs under a project file such as the following:

```
/* PROJECT FILE */
progname.c(def.h,scsi.h)
scsilib.lib
/****************/
```

'Progname' is the name of the program being edited. The files "def.h" and "scsi.h" define the system, while the 'scsilib' statement provides linkage to the library of prefabricated functions. This project file must be specified in the 'project' and 'primary c file' of the compiler's environment. TURBO C's parameters must also be configured as follows;

```
Compiler:  Model:                SMALL
           Optimization for:     SPEED

Linker:    Case sensitive link:  OFF
```

All other parameters: ON

The SCSI Command System installation instructions also review TURBO C's needs for proper usage.

Microsoft C users, after standard 'msc progname.c' compilation, need only to answer the linker's prompt of 'Libraries[.LIB]=' with 'scsilib.lib'. Drive specification of the library must be properly specified. This should eliminate all unresolved externals and allow proper linkage. Installation instructions for MSC also provide this information regarding main programs.

2.4 U P P E R L E V E L P R O G R A M M I N G

The term 'upper level' implies usage of the provided library of functions. The function calling level of system use is simple. Programs are short, quick and powerful. In addition, very little knowledge of 'C' language is needed to operate the system in this way.

As in all programs that use the library it is necessary to include "scsi.h" and "def.h". Establishing a buffer is also an important step to remember at this level of usage. In order to send a command to the proper controller the target identification and logical unit number must be set. In regard to establishing a buffer and setting the target/unit there are simple to use functions, as demonstrated in the examples below.

The examples below use many functions, but do not describe their syntax. All functions are described in the 'Function Library' portion of text.

NOTE: All examples are in 'C' language.

2.4.1 DISCUSSION OF UPPER LEVEL EXAMPLES

1) Example number one is the 'classic' read example. As a first step, the two files, "scsi.h" and "def.h" are included. Remember that this step is essential. After a buffer is established by calling 'setbuf', the target i.d. and logical unit number are set. Setting the 'tid/lun' creates a communication path between initiator and target. The start medium rotation command is sent. A read of one logical sector takes place, sector 203 hex. After the read, the drive is stopped, the buffer is displayed, a disk copy is made, and a hard copy is created.

```
/*********************************************************/
/* example program #1                                    */
/* the classic read example                              */
/*                                                       */
/*                                                       */
```

```
#include "scsi.h"          /* must be included                */
#include "def.h"           /* must be included                */

main()
{
setbufd(1024);            /* establish a 1024 byte buffer    */
tidlu(0,0);               /* target=0 logical unit=0         */
sarsto(1);                /* start the drive                 */
sread(1,0x203L);          /* read logical sector 203hex, 1K  */
sarsto(0);                /* stop the drive                  */
displbu();                /* display the buffer on the screen*/
makebuf("b:data.dat");    /* make a disk copy of the buffer  */
prinbuf();                /* print a buffer hard copy        */
}                         /* end of program                  */
/*                                                            */
/************************************************************/
```

2) Example number two is a write example. After including the proper files, the main program is opened and a buffer is created. The device is then selected using the 'tidlu' function. The program buffer for the write is filled from two sources. The first data source is a disk file; it fills the first 512 bytes of the 1K buffer. The other 1/2K is filled from the 'userbuf' command. 'Userbuf' will ask the user for 16 bytes of information. This pattern of 16 bytes will be repeated for the remaining 1/2K, thus filling the remainder of the buffer. After the drive is spun-up, the write takes place. 'Erdec' performs a request sense and decodes the data it receives. It will respond with the proper information in the event that an error has occurred. When the command is completed the drive is stopped.

```
/************************************************************/
/* example program #2                                       */
/* the classic write example                                */

#include "scsi.h"          /* must be included                */
#include "def.h"           /* must be included                */

main()
{
setbufd(1024);            /* establish a 1024 byte buffer    */
tidlu(0,0);               /* target=0 logical unit=0         */
loadbuf("b:data.dat",512,0);/*load 1/2k from disk file       */
userbuf(16,512,512);      /* 16 byte pat for the other 1/2k  */
sarsto(1);                /* start the drive                 */
swrit(1,0x204L);          /* write to logic sect 204hex, 1K  */
erdec();                  /* check for errors                */
sarsto(0);                /* stop the drive                  */
}                         /* end of program                  */
/*                                                            */
/************************************************************/
```

3) Example number three executes a mode selection command. It demonstrates how the 'userbuf' function can be used to fill the buffer with the mode selection data through user interaction. Again, the program starts as the last two examples do. The 'userbuf' statement asks that a six byte pattern be entered by the user. The pattern is then repeated in bytes 0 to 5. 'Userbuf' will prompt the user for those six bytes of data required for a mode select. The 'mosel' command selects the mode according to this data in the buffer.

```
/*********************************************************/
/* example program #3                                    */
/* the buffer as a parameter list                        */

#include "scsi.h"          /* must be included           */
#include "def.h"           /* must be included           */

main()
{
setbufd(6);       /* establish a 6 byte buf              */
tidlu(0,0);       /* target=0 logical unit=0             */
userbuf(6,6,0);   /* user enters the mode select dat     */
mosel(6);         /* param list of 6 bytes in buffer     */
/*                                                       */
/* EXPLANATION:  The mode selection data is input by the user by
using the 'userbuf' function. The user enters the data describing
the mode desired for selection. That data, now in the program
buffer, is used as the parameter list for the mode selection. All
functions that need a data list will expect the data to be in the
program's buffer before the command is called.
                  /*                                   */
}                  /* end of program                   */
/*********************************************************/
```

4) The last example performs a mode sense. The six bytes of data are returned to the program buffer, where they can be displayed or saved as any other data in the buffer.

```
/*********************************************************/
/* example program #4                                    */
/* the buffer allocation                                 */

#include "scsi.h"          /* must be included           */
#include "def.h"           /* must be included           */

main()
{
setbufd(6);              /* establish a 6 byte buffer    */
tidlu(0,0);              /* target=0 logical unit=0      */
mosen(6);                /* 6 bytes of data are to be sensed*/
bufnum(1,6);             /* display the returned mode data  */
```

```
/* EXPLANATION:  The mode selection data is returned to the
program's buffer. The data can then be displayed. All SCSI
functions that return data will do so in the program's buffer.
                                                              */
}                              /*    end of program           */
/*************************************************************/
```

Note that the above examples use many of the library functions, but do not specifically describe them. If further information regarding a function is needed, refer to the second section of the user's guide. All functions are completely described there.

2.5 LOWER LEVEL PROGRAMMING

As discussed before, there are two essential structures to this software; the buffer structure, and the command block structure. These variables can be modified in a main program, or a function can be created based on them. This type of programming is referred to as lower level programming. The two structures are defined in 'C' language as follows:

```
struct{
       unsigned char by[10240]; /* the 10K buffer      */
       }buffer;

struct{
       unsigned char cmd; /* the command driver byte   */
       unsigned char tidlun; /* target id/ logical unit*/
       unsigned char cdb[12];/* the command block      */
       unsigned char scsirc; /* the driver return code */
       unsigned char targrc; /* the target return code */
       unsigned int bufseg;  /* the buffer segment     */
       unsigned int bufoff;  /* the buffer offset      */
       unsigned int buflen;  /* the buffer length      */
       }cmdblk;              /* the structure's name   */

/*   NOTE: unsigned char = 1 byte = 8 bits
           unsigned int  = 2 bytes = 16 bits   */
```

The buffer structure represents the program's buffer, whose maximum size is 10K. Any element in the buffer can be referred to by its number in the structure, for instance 'buffer.by[122]' is the one hundred twenty second byte, eight bit quantity, of the buffer. This is especially useful when displaying the entire buffer is not desirable, or when buffer information is needed for some other purpose. There are 10240 possible bytes, each representing a unique byte of data. Within main programs these variables can be accessed by their unique names to examine the data that they represent.

The cmdblk structure is somewhat more complicated. The first byte, cmd, is the command driver byte and can be either 00 or 60

hex. Hex 00 implies that a command will follow while 60 hex implies a bus reset. Byte two, tidlun, is the byte containing information about the target id and logical unit number. In all cases, this byte can properly be set up using the 'tidlu' library function. The next thirteen bytes describe the command block. The command data should be installed here. The next two bytes describe the driver return code and target return code. They are used to indicate erroneous command transfer or execution. The 'erdec' routine has been provided in the function library to eliminate the need to deal with these two return codes. 'Erdec' decodes errors and displays their meaning. The last three structure members describe the buffer's segment, offset and length in memory. These are set up by using the 'setbufd' command.

The variables primary concern are those representing the buffer, and the 13 bytes which describe the actual command data in the command block. Further documentation on the command block structure can be found in the Micro Design International Host Adapter Manual. This system follows the Micro Design command block format exclusively.

2.5.1 LOWER LEVEL FUNCTION CREATION EXAMPLE

This example illustrates use of the program's variable structures in a main program and then converts that main program into functions. These functions, when in object form, can be permanently installed in a library or linked with main programs. This example covers the steps required to achieve this. Note that two library functions which are very useful in this type of programming are used in these examples. They are called 'clear' and 'cif', and are discussed in the function library portion of the text.

The first part of the example shows command block modification and buffer access through variable reference. The command block modification sets up the normal SCSI start command. The ANSI SCSI manual can be referred to, confirming this command's syntax. The assembly language routine 'cif' is then called to send the command. With the drive started, and a buffer set up, a library short read command is performed. With the data read into the buffer, access through variable name is used to produce a byte by byte buffer display. This display is unlike 'displbu', which displays in a special format (a 16 byte by 16 byte display). Note that the limit on the loop is the buffer length element of command block structure. This is an example of a main program with lower level characteristics and is the first step in the creation of a new function.

```
/************STEP NUMBER ONE************/
#include "scsi.h"
#include "def.h"    /* INCLUDE AS NORMAL*/
main()
{
int i;
i=0;                    /* initialize the counter */
setbufd(1024);          /* set up a 1K buffer     */
tidlu(0,0);             /* set the target/ lun    */
clear();                /* clear the command block*/
cmdblk.cdb[0]=0x1b; /* the start/stop command   */
cmdblk.cdb[4]=1;        /* the start bit          */
cif(&cmdblk);           /* drive the command      */
sread(1,0x200L);        /* a short read of 200h   */
                        /* display buffer bytes as*/
                        /* a column. note the use */
                        /* of the variable buflen */
while (i<= cmdblk.buflen) {
printf("#%d...%x\n",i,buffer.by[i]);
i++;
}
}
```

To make a 'byte display' and 'unit start' object function modules, we must first isolate them from the main program. It is helpful to try functions in their non-library form to insure that they are working properly. As seen in step two, the main program calls the functions still included below in the source code. This step can eliminate a lot of parameter passing problems before the object file library stages of function creation.

```
/********* STEP NUMBER 2**********/
#include "scsi.h"
#include "def.h"        /* include both files      */
main()
{
tidlu(0,0);
start();                /* the new 'unit start'    */
setbufd(1024);
sread(1,0x200L);
byter();                /* the new 'byte display'  */
}
/******************/
start()
{
clear();                /* clear the command block */
cmdblk.cdb[0]=0x1b;     /* set up the start command*/
cmdblk.cdb[4]=1;        /* start bit               */
cif(&cmdblk);           /* send the start command  */
return;
}
/******************/
```

```
/*******************/
byter()
{
int i;
i=0;
while (i<= cmdblk.buflen) /* byte display        */
{
printf("byte number%d...%x\n",i,buffer.by[i]);
i++;
}
/*********************/
return;
}
```

Having established that the functions do indeed work, they can now be isolated from the main program. Note that the function 'byter' contains only "def.h". This is because 'byter' user no functions in the library: it needs only the variables defined. The functions are now ready to be compiled into an object form. The object files created are now of a useful form. To be used by a main program, they need only to be added to the main program by the linker. As a further step, a library manager control command which adds new modules to the existing library can be executed. For the Microsoft Library Manager it is simple. Use the library line command 'lib scsilib.lib + name of the new object module'. Turbo C's library manager command is similar.

```
/* F I L E   O N E                                  */
/* new start function to be added to the lib */
#include "def.h"   /* include these files    */
#include "scsi.h"  /*                        */
/*******************/
start()
{
clear();
cmdblk.cdb[0]=0x1b;
cmdblk.cdb[4]=1;
cif(&cmdblk);
return;
}
/*******************/
```

```
/* F I L E    T W O                                 */
/* byte display function to be added to lib  */
/* or linked to by main programs             */
#include "def.h"   /* include only this file */
byter()
{
```

```
int i;
i=0;
while (i<= cmdblk.buflen) {
printf("byte number %d...%x\n",i,buffer.by[i]);
i++;
}
return;
}
```

A new main program can now be written to test the object versions of the new functions. The last part of this example represents this program. When linked to the new object file or files, it should test the newly created functions. If the new functions have been added to the library then normal library linkage should eliminate all unresolved externals. The functions generated in the above example are members of the library and were originally added by use of the procedure involving the Microsoft Library Manager. Later they were added to the main module of code. These functions can be renamed and reused to practice low level programming and function creation skills.

```
/* the new main program to call library */
/* or to link object modules to         */
#include "scsi.h"
#include "def.h"

main()
{
tidlu(0,0);
start();
setbufd(1024);
sread(1,0x200L);
byter();
}
```

2.5.2 FUNCTION EXAMPLE FROM THE LIBRARY

As a further example of a low level function, a segment of code describing the library's short write is provided:

```
/*************************************************************/
/* S H O R T   W R I T E   F U N C T I O N                   */
/*************************************************************/

swrit(len,addr)
unsigned long addr; /* receives the address                  */
byte len;           /* receives the length                   */
{
/* function assumes a buffer is set up                       */
/* function assumes the buffer is full of data               */
/* function assumes the target/logical unit are set          */
```

```
/* function simply sends the write command                    */
byte ah,am,al;
clear();
ah=mh(addr);                /* break the long int into        */
am=ml(addr);                /* its separate bytes             */
al=lo(addr);                /*                                */
cmdblk.cdb[0]=0x0A;         /* the short write command        */
cmdblk.cdb[1]=(cmdblk.cdb[1] & 0xe0) | ah; /*                  */
                            /* mask&tid/lun or addr high*/
cmdblk.cdb[2]=am;           /* the middle low addr byte       */
cmdblk.cdb[3]=al;           /* the lower addr byte            */
cmdblk.cdb[4]=len;          /* length                         */
return(cif(&cmdblk));       /* send the command               */
}
```

This function is sent the length of the write which is to occur and the address at which to write. The long unsigned integer representing the address is broken into its separate bytes by using library functions. The command block is set up for the short write. Note that cmdblk.cdb[1] requires some logic. The lun is needed in the top 3 bits and is assumed to be set up as so. A bit mask prevents extraneous data from being entered in the command. This byte is then 'or'ed with the high byte of address. Finally the call to the command interfacing function is made to send the command. Note that this section of code contains no include files, that is because all definitions necessary for its use occur elsewhere in the same module of code. See also the function library portion of text; 'swrit' is described from a usage point of view there.

2.5.3 LOWER LEVEL BUFFER EXAMPLE

The following example is a function that allows a mode selection. It places data in the buffer for a mode selection, then executes that mode selection with a library command.

```
/*************************************************************/
/* S P E C I A L   M O D E   S E L E C T   F U N C T I O N   */
/*************************************************************/

#include "scsi.h"
#include "def.h"
spmode()
{
/* function assumes a buffer is set up                        */
/* function assumes target and logical unit are set           */
/*                                                            */
buffer.by[0]= 0x80;     /* placing mode data in the           */
buffer.by[1]= 0x12;     /* first six bytes of the             */
buffer.by[2]= 0;        /* program buffer                     */
buffer.by[3]= 0xff;     /*                                    */
buffer.by[4]= 0x0a;     /*                                    */
```

```
buffer.by[5]= 0;        /*                                  */
mosel(6);               /* call for a mode select with      */
                        /* the buffer data                  */
return;
}
```

Though the library provides data manipulating capabilities it is often more convenient to interact directly with the buffer variables. This mode selection is a much cleaner approach than upper level example number three. This routine is much more flexible and requires only a knowledge of the buffer structure to be written. Additionally, it can be added to the set of library functions to further expand system capabilities.

USER'S GUIDE

section two

USER'S GUIDE: SECTION TWO

2.6 THE FUNCTION LIBRARY

The function library portion of text discusses each function that is available for use in the system. It describes each with the following format:

NAME

FUNCTION NAME: returned name(parameters)

FUNCTION TYPE: ATG,SONY,SCSI, or TOOL

PARAMETERS PASSED: name==>type
name==>type
name==>type

PARAMETERS RETURNED: type

PURPOSE:

REQUIREMENTS:

SPECIAL:

EXAMPLE:

RETURNS:

The function type can be SONY, ATG,SCSI or TOOL. A SCSI type function implies that it is general and can be used with any SCSI device. A TOOL function works within the program and usually directs, modifies or supplies data. TOOL types do not affect SCSI devices or communicate directly with them. For each function a purpose is discussed. The purpose describes the result of executing the command. The requirements subsection of each function description repeats the type of parameters and provides a definition for each. If the parameters have a limited set of values the range is presented here. Special is a section that is intended to eliminate program errors by presenting the 'tricks', if you will, to each function. It should give you a better chance of using each function without encountering difficulty. Examples of how the function call will look in a 'C' language program are provided. Comments on the example's code describe the result of that line at run time. The values returned by the function, if any, are examined lastly. The definitions of the values returned are given. Each function description is intended to describe purpose, parameters and proper use.

The functions appear in this section in alphabetized order as follows.....

NAME	TYPE	DESCRIPTION
ascibuf	TOOL	displays buffer as ascii
bufnum	TOOL	displays a buffer section
byter	TOOL	displays buffer byte by byte
CIF	SCSI	command interface function
clear	TOOL	clears the command block
displbu	TOOL	displays the entire buffer
dread	ATG	diagnostic read
dwrit	ATG	diagnostic write
edsene	SONY	eject sense toggle
erdec	TOOL	error decoder
fillbuf	TOOL	fills the buffer with 1 byte
firbl	ATG	first blank sector search
hi	TOOL	returns the long's high byte
inqir	SCSI	the inquiry function
lo	TOOL	returns the long's low byte
loadbuf	TOOL	loads the buffer from disk
makebuf	TOOL	creates a buffer copy on disk
mh	TOOL	returns the middle high byte
ml	TOOL	returns the middle low byte
mosel	SCSI	mode selection function
mosen	SCSI	mode sense function
prealo	ATG	medium removal toggle
prinbuf	TOOL	print the buffer's contents
readb	ATG	read the drives buffer
readc	ATG	read the drive capacity
readdr	ATG	read data redundancies
readid	ATG	read the disk id
readl	SCSI	long read command
recodw	SONY	recover disk warning
rese	SCSI	reset of the SCSI bus
resen	SCSI	request sense
rodis	ATG	read the ODI status
rzzo	SCSI	rezero the drive
saiifo	SONY	sense alternate information
sarsto	SCSI	start/stop toggle
scopy	SONY	the SONY copy command
scssix	SCSI	user generated command
scsten	SCSI	user generated command
scstwl	SCSI	user generated command
sdisej	SONY	disk eject command
seekbs	SONY	blank sector search
seekl	SCSI	long seek command
seekws	ATG	written sector search
setbufd	TOOL	set up the data buffer
setmp	ATG	set medium parameters
smove	SONY	the move command
srdres	SONY	receive diagnostics

sread	SCSI	short read
srele	SONY	release command
srese	SONY	reserve command
ssdia	SONY	send diagnostics
sseek	SCSI	short seek
start	SCSI	starts the medium
swrit	SCSI	short write
tesur	SCSI	test unit ready
tidlu	TOOL	set the target/lun numbers
userbuf	TOOL	user generated buffer
verif	ATG	verify command
writeb	ATG	write to the drive buffer
writl	SCSI	the long write command
wrver	ATG	write and verify command

FUNCTION NAME: ascibuf()

FUNCTION TYPE: TOOL

PARAMETERS PASSED: none

PARAMETERS RETURNED: none

PURPOSE: Ascibuf dumps the contents of the buffer as ascii characters to the screen. If the buffer is 1K it displays that 1K. Ascibuf will always display the entire buffer as characters.

REQUIREMENTS: none

SPECIAL: The entire buffer will be dumped, the size of the dump is the size of the buffer.

EXAMPLE:

```
ascibuf();    /* displays the buffer as char*/
```

FUNCTION NAME:	bufnum(start,stop)
FUNCTION TYPE:	TOOL
PARAMETERS PASSED:	start ==> integer stop ==> integer
PARAMETERS RETURNED:	none

PURPOSE: Bufnum displays the buffer from the byte number specified by start to the byte number specified by stop.

REQUIREMENTS: Start and stop are integers representing the byte to start the display and the byte to end the display.

SPECIAL: If start and stop are equal only that specific byte will be displayed.

EXAMPLE:

```
bufnum(0,6);          /* display bytes 0 to 6 */

bufnum(12,12);        /* displays only byte 12*/
```

FUNCTION NAME: byter()

FUNCTION TYPE: TOOL

PARAMETERS PASSED: none

PARAMETERS RETURNED: none

PURPOSE: Byter is the byte displaying subroutine that is developed and added to the library in the lower level program example. Byter displays the entire buffer in a column form, presenting each byte and its number.

REQUIREMENTS: none

SPECIAL: The entire buffer will be displayed.

EXAMPLE:

```
byter();        /* displays the buffer */
```

FUNCTION NAME:	int	cif(&cmdblk)
FUNCTION TYPE:		SCSI
PARAMETERS PASSED:		pointer to the command block structure
PARAMETERS RETURNED:		int

PURPOSE: CIF is the assembly language routine that drives the command. It sends the pointer to the command block to the ROM on the host adapter car and the command is placed on the SCSI bus for the proper target/lun.

REQUIREMENTS: CIF must send a pointer to a command block structure

SPECIAL: The syntax shown below is proper for pointing to the library's command block. It should be use in this way to drive all lower level programming commands.

EXAMPLE:

```
cif(&cmdblk); /* drive the command */
```

RETURNS: One integer is returned. It is used to indicate error. If the returned value is equal to FFh then an error has occurred in the execution of the command. 'Erdec' can be called to decode the error

FUNCTION NAME: clear()

FUNCTION TYPE: TOOL

PARAMETERS PASSED: none

PARAMETERS RETURNED: none

PURPOSE: Clear initializes the command block structure. Clear does not effect the buffer parameters set with "setbufd", or the target id and logical unit numbers set with "tidlu".

REQUIREMENTS: none

SPECIAL: none

EXAMPLE:

```
clear();          /* clears the command block */
```

FUNCTION NAME: displbu()

FUNCTION TYPE: TOOL

PARAMETERS PASSED: none

PARAMETERS RETURNED: none

PURPOSE: Displbu dumps the contents of the buffer to the screen in a special 16 byte format. If the buffer is 1K it displays that 1K. Displbu will always display the entire buffer.

REQUIREMENTS: none

SPECIAL: The entire buffer will be displayed.

EXAMPLE:

```
displbu();          /* displays the buffer */
```

FUNCTION NAME: int dread(len,addr)

FUNCTION TYPE: ATG

PARAMETERS PASSED: unsigned int ==> len
long unsigned int ==> addr

PARAMETERS RETURNED: int

PURPOSE: Dread performs the diagnostic read command when the drive is in the diagnostic mode. Dread reads user data but also obtains additional information such as edac code. Because this additional data is read the buffer must be of a size to accommodate sectors of 1115 bytes in length.

REQUIREMENTS: Len represents the number of logical sectors to be read, or the length of data transfer. Addr represents the address to start the reading of data. Three bytes of address are allowed.

SPECIAL: The drive must be put into diagnostic mode before this command can be used. It is also important to remember that sectors read include additional information besides user data. Each sector is 1115 bytes in size

EXAMPLE:

```
dread(1,0x205L); /* read 1115bytes from 205h*/

dread(10,1L);    /* read ten sectors starting
                    at 1 */
```

RETURNS: One integer is returned. It is used to indicate error. If the returned value is equal to FFh then an error has occurred in the execution of the command. 'Erdec' can be called to decode the error.

FUNCTION NAME:	int	dwrit(len,addr)
FUNCTION TYPE:		SCSI
PARAMETERS PASSED:		unsigned int ==> len long unsigned int ==> addr
PARAMETERS RETURNED:		int

PURPOSE: Dwrit performs the diagnostic read command when the drive is in the diagnostic mode. Dwrit writes data in frames of 1115 bytes without edac error correction.

REQUIREMENTS: Len represents the number of logical sectors to be written, or the length of data transfer. Addr represents the address to start the writing of data. Three bytes of address are allowed.

SPECIAL: The drive must be put into diagnostic mode before this command can be used. It is also important to remember that sectors size is 1115 bytes.

EXAMPLE:

```
dwrit(1,0x205L); /* write 1115bytes to 205h*/

dwrit(10,1L);    /* write ten sectors starting
                    at 1 */
```

RETURNS: One integer is returned. It is used to indicate error. If the returned value is equal to FFh then an error has occurred in the execution of the command. 'Erdec' can be called to decode the error.

FUNCTION NAME:	int	edsene(flag)
FUNCTION TYPE:		SONY
PARAMETERS PASSED:		unsigned int ==> flag
PARAMETERS RETURNED:		int

PURPOSE: This command prevents or allows the sensing of the disk eject command.

REQUIREMENTS: Flag must be an unsigned integer. If flag is equal to 1 then the drive is enabled to sense the eject command. If flag is equal to 0 then the eject command has been disabled, the drive is now unable to sense the command.

SPECIAL: Flag must be only 1 or 0.

EXAMPLE:

```
edsene(1)  /*eject sense is enabled*/

edsene(0)  /*eject sense is disabled*/
```

RETURNS: One integer is returned. It is used to indicate error. If the returned value is equal to FFh then an error has occurred in the execution of the command. 'Erdec' can be called to decode the error.

FUNCTION NAME: erdec()

FUNCTION TYPE: TOOL

PARAMETERS PASSED: none

PARAMETERS RETURNED: none

PURPOSE: Erdec performs a request sense command then decodes the data received and finally prints out the error that occurred.

REQUIREMENTS: none

SPECIAL: This command should be used in conjunction with the returned value indicating error that most of the functions feature. If the returned value of a command is FFh then erdec can be used to find out what happened.

EXAMPLE:

```
erdec();          /* decodes sense info */
```

FUNCTION NAME:	fillbuf(element,fill,start)
FUNCTION TYPE:	TOOL
PARAMETERS PASSED:	unsigned char ==> element unsigned int ==> fill unsigned int ==> start
PARAMETERS RETURNED:	none

PURPOSE: Fillbuf is a subprogram that repeats element for the desired length in the program buffer. Fillbuf fills the buffer with element.

REQUIREMENTS: Element is the byte to repeat in the buffer, it should be a single byte of information. Fill is an unsigned integer representing how much of this pattern should be placed in the buffer. Start is also an unsigned integer, it depicts where in the buffer the data will begin.

SPECIAL: The total fill size and the start value should not exceed the upper boundary of the program buffer.

EXAMPLE:

```
fillbuf(02,1024,0);     /*fills the buffer with 1K of
                          02*/

fillbuf(0xff,512,152); /*fills a 1/2K space in the
                         buffer starting at the 152th
                         byte with 0xff. */
```

FUNCTION NAME:	int	firbl(nos,addr,rel)
FUNCTION TYPE:		ATG
PARAMETERS PASSED:		long unsigned int ==> nos long unsigned int ==> addr unsigned int ==> rel
PARAMETERS RETURNED:		int

PURPOSE: Firbl performs a search for the first blank sector. It returns the address of this sector in the buffer.

REQUIREMENTS: The number of sectors, nos, must be a long unsigned integer. Only the low two bytes of this value will be used. Addr is the variable that specifies the 4 byte address. Rel is used to implement the relative addressing mode. (1=relative addressing).

SPECIAL: The address of the first blank sector found in the search will be returned in the buffer. It is important to properly set up the buffer before use.(see buffer examples)

EXAMPLE:

```
firbl(10L,204L,0); /*search 10sect. starting */
                   /*at address 204          */
```

RETURNS: One integer is returned. It is used to indicate error. If the returned value is equal to FFh then an error has occurred in the execution of the command. 'Erdec' can be called to decode the error.

FUNCTION NAME: unsigned int hi(luint)

FUNCTION TYPE: TOOL

PARAMETERS PASSED: long unsigned int ==> luint

PARAMETERS RETURNED: unsigned int

PURPOSE: This function is used to return the highest byte of a long unsigned integer.

REQUIREMENTS: The value you send to this routine must be a long unsigned integer.

SPECIAL: none.

EXAMPLE:

```
hi(0x10203040L);  /*will return 10*/
```

RETURNS: One unsigned integer is returned. It is the highest byte of the long unsigned integer which was sent.

FUNCTION NAME: int inqir (allo)

FUNCTION TYPE: SCSI

PARAMETERS PASSED: unsigned integer ==> allo

PARAMETERS RETURNED: int

PURPOSE: Inqir performs the SCSI inquiry function.

REQUIREMENTS: Allocation is an unsigned integer between 0 and 255 specifying the allocation length of the inquiry.

SPECIAL: Allocation should by between 0 and 255.

EXAMPLE:

```
inqir(0);       /*zero allocation length*/

inqir(255);     /*255 allocation length */
```

RETURNS: One integer is returned. It is used to indicate error. If the returned value is equal to FFh then an error has occurred in the execution of the command. 'Erdec' can be called to decode the error.

FUNCTION NAME: unsigned int lo(luint)

FUNCTION TYPE: TOOL

PARAMETERS PASSED: long unsigned integer==>luint

PARAMETERS RETURNED: unsigned int

PURPOSE: This function is used to return the low byte of a long unsigned integer.

REQUIREMENTS: The value you send to this routine must be a long unsigned integer.

SPECIAL: none.

EXAMPLE:

```
lo(0x10203040L);  /*will return 40*/
```

RETURNS: One unsigned integer is returned. It is the low byte of the long unsigned integer which was sent.

FUNCTION NAME:	int	loadbuf(name,length,start)
FUNCTION TYPE:		TOOL
PARAMETERS PASSED:		char[20] ==> name unsigned int ==> length unsigned int ==> start
PARAMETERS RETURNED:		int

PURPOSE: Loadbuf loads a data file in the standard format into the program buffer.

REQUIREMENTS: Name is a character string up to 20 characters long which specifies the drive and name of the data file to be created. Length specifies the length of data to be read from the file and placed in the program buffer. Start specifies where the data should begin in the buffer.

SPECIAL: The file read from disk must be in the standard format, the format that the 'makebuf' command creates. The upper boundary on the programs data buffer should not be exceeded.

EXAMPLE:

```
loadbuf("b:random.dat",1024,0); /*loads 1K from
                                  random.dat in the
                                  'b' drive and starts
                                  at the beginning of
                                  the program buffer.*/


loadbuf("b:random.dat",512,512;/*loads 1/2K from
                               random.dat in the
                               'b' drive and starts at
                               the 512th byte of the
                               program buffer.*/
```

RETURNS: One integer is returned. It is used to indicate error. If the returned value is equal to FFh then an error has occurred in the execution of the command. 'Erdec' can be called to decode the error.

FUNCTION NAME: int makebuf(name)

FUNCTION TYPE: TOOL

PARAMETERS PASSED: char[20] ==> name

PARAMETERS RETURNED: int

PURPOSE: Makebuf takes the current program buffer and creates a disk file of it in the standard format.

REQUIREMENTS: Name is a character string 20 characters long which specifies the drive and name of the data file to be created.

SPECIAL: The entire buffer will be placed in the disk file. This file is in the proper format for reloading with the 'loadbuf' command.

EXAMPLE:

```
makebuf("b:random.dat"); /*makes a disk file called
                           random.dat of the current
                           program buffer.*/
```

RETURNS: One integer is returned. It is used to indicate error. If the returned value is equal to FFh then an error has occurred in the execution of the command. 'Erdec' can be called to decode the error.

FUNCTION NAME: unsigned int mh(luint)

FUNCTION TYPE: TOOL

PARAMETERS PASSED: long unsigned integer ==> luint

PARAMETERS RETURNED: unsigned int

PURPOSE: This function is used to return the middle high byte of a long unsigned integer.

REQUIREMENTS: The value you send to this routine must be a long unsigned integer.

SPECIAL: none.

EXAMPLE:

```
mh(0x10203040L)  /*will return 20*/
```

RETURNS: One unsigned integer is returned. It is the middle high byte of the long unsigned integer which was sent.

FUNCTION NAME: unsigned int ml(luint)

FUNCTION TYPE: TOOL

PARAMETERS PASSED: long unsigned integer ==> luint

PARAMETERS RETURNED: unsigned int

PURPOSE: This function is used to return the middle low byte of a long unsigned integer.

REQUIREMENTS: The value you send to this routine must be a long unsigned integer.

SPECIAL: none.

EXAMPLE:

```
ml(0x10203040L);  /*will return 30*/
```

RETURNS: One unsigned integer is returned. It is the middle low byte of the long unsigned integer which was sent.

FUNCTION NAME:	int	mosel(pll)
FUNCTION TYPE:		SCSI
PARAMETERS PASSED:		unsigned integer ==> pll
PARAMETERS RETURNED:		int

PURPOSE: Mosel performs the SCSI mode selection function. It transfers mode select data to the target during the data out phase. The mode selection data to be transferred should be stored in the buffer before this functional call.

REQUIREMENTS: Pll is an unsigned integer between 0 and 255 specifying the parameter list length for the selection of mode.

SPECIAL: Pll should by between 0 and 255. It is usually set to 6. Before this command can be executed mode selection data must be entered into the buffer. This can be done with the userbuf function.(See buffer use examples).

EXAMPLE:

```
mosel(0);       /*zero param. list  length*/

mosel(6);       /*6 allocation length */
```

RETURNS: One integer is returned. It is used to indicate error. If the returned value is equal to FFh then an error has occurred in the execution of the command. 'Erdec' can be called to decode the error.

FUNCTION NAME:	int	mosen(allo)
FUNCTION TYPE:		SCSI
PARAMETERS PASSED:		unsigned integer ==> allo
PARAMETERS RETURNED:		int

PURPOSE: Mosen performs the SCSI mode sense function. Data is transferred from the target regarding the device parameters.

REQUIREMENTS: Allo is an unsigned integer between 0 and 255 specifying the allocation length provided for the information being returned. The data returned by this function is stored in the program buffer. The allocated length should not exceed the buffer size.

SPECIAL: Allo should be between 0 and 255. It is usually set to 6. Before this command can be executed the buffer must be set to at least the size allocated. The data returned with this command is placed in the buffer.(see buffer examples).

EXAMPLE:

```
mosen(0);       /*senses 0 bytes of data*/

mosen(6);       /*senses 6 bytes of data*/
```

RETURNS: One integer is returned. It is used to indicate error. If the returned value is equal to FFh then an error has occurred in the execution of the command. 'Erdec' can be called to decode the error.

FUNCTION NAME:	int	prealo (flag)
FUNCTION TYPE:		ATG
PARAMETERS PASSED:		unsigned integer ==> flag
PARAMETERS RETURNED:		int

PURPOSE: Prealo performs the prevent/allow medium removal function. It also enables or disables the front panel drive controls.

REQUIREMENTS: Flag is an unsigned integer and should be only a 1 or a 0. A 1 indicates prevent medium removal/disable front panel, and a 0 indicates allow medium removal/enable front panel.

SPECIAL: Flag should be only 1 or 0.

EXAMPLE:

```
prealo(1);      /* prevents medium removal */

prealo(0);      /* allows medium removal   */
```

RETURNS: One integer is returned. It is used to indicate error. If the returned value is equal to FFh then an error has occurred in the execution of the command. 'Erdec' can be called to decode the error.

FUNCTION NAME: int prinbuf()

FUNCTION TYPE: TOOL

PARAMETERS PASSED: none

PARAMETERS RETURNED: int

PURPOSE: Prinbuf dumps the contents of the buffer to the printer.If the buffer is 1K it prints that 1K. Prinbuf will always print the entire buffer.

REQUIREMENTS: none

SPECIAL: The entire buffer will be dumped; the size of the dump is the size of the buffer.

EXAMPLE:

```
prinbuf();              /* prints out the buffer*/
```

RETURNS: One integer is returned. It is used to indicate error. If the returned value is equal to FFh then an error has occurred in the execution of the command. 'Erdec' can be called to decode the error.

FUNCTION NAME:	int	readb(tl,bn)
FUNCTION TYPE:		ATG
PARAMETERS PASSED:		unsigned int ==> tl unsigned int ==> bn
PARAMETERS RETURNED:		int

PURPOSE: Readb reads data from the specified scsi device drive buffer.

REQUIREMENTS: Tl can be one or zero. When tl=0 read length is 1024 bytes. When tl=1 the command reads 1115 bytes from the drive buffer. Bn is the buffer number, 0 corresponds to drive buffer 0 and 1 corresponds to drive buffer 1.

SPECIAL: The drive must be put into diagnostic mode before this command can be used. It is also important to remember that sectors size can be 1115 bytes in size the buffer should be able to accommodate this.

EXAMPLE:

```
readb(1,1);/*read 1115 bytes from drive buffer 1*/

readb(0,0);/*read 1K from drive buffer 0        */
```

RETURNS: One integer is returned. It is used to indicate error. If the returned value is equal to FFh then an error has occurred in the execution of the command. 'Erdec' can be called to decode the error.

FUNCTION NAME:	int	readc(addr,rel)
FUNCTION TYPE:		ATG
PARAMETERS PASSED:		long unsigned int ==> addr unsigned int ==> rel
PARAMETERS RETURNED:		int

PURPOSE: Readc performs the SCSI read capacity command which request that information regarding the capacity of the logical unit be returned to the initiator.

REQUIREMENTS: Addr is a long unsigned integer describing the logical sector address for the capacity check. Rel is a 1 or 0 to indicate relative addressing or normal addressing.(1=relative)

SPECIAL: The address is specified to be 4 bytes long truncation of oversized long integers will occur. Before this command can occur there must be established a buffer for the read capacity data to be transferred into. The buffer can be setup using setbufd.

EXAMPLES:

```
readc(00L,0)      /*capacity of 0 without rel.*/

readc(1024L,1)    /*capacity of relative 1024 */
```

RETURNS: One integer is returned. It is used to indicate error. If the returned value is equal to FFh then an error has occurred in the execution of the command. 'Erdec' can be called to decode the error.

FUNCTION NAME:	int	readdr(len,addr)
FUNCTION TYPE:		ATG
PARAMETERS PASSED:		unsigned int ==> len long unsigned int ==> addr
PARAMETERS RETURNED:		int

PURPOSE: Readdr performs the read data and redundancies command, an ATG diagnostic function.

REQUIREMENTS: Len is an unsigned integer representing the length of the read to occur and is in the units of logical sectors. Addr is the address of the sector at which to begin reading.

SPECIAL: A buffer must be established to execute this command and the length of the read should not exceed the buffer length. The drive must be in diagnostic mode before this command can be executed.

EXAMPLE:

```
readdr(1,6L);          /* read the dat and redun of
                          sector six*/
```

RETURNS: One integer is returned. It is used to indicate error. If the returned value is equal to FFh then an error has occurred in the execution of the command. 'Erdec' can be called to decode the error.

FUNCTION NAME: int readid()

FUNCTION TYPE: SONY

PARAMETERS PASSED: none

PARAMETERS RETURNED: int

PURPOSE: This reads the disk id of the current specified disk.

REQUIREMENTS: There must be at least a 1K buffer established before this command can take place. It will always return 1k of data from the shortest radius of the disk.

SPECIAL: This command needs a 1K buffer.

EXAMPLE:

```
readid()  /*reads disk id (1k)*/
```

RETURNS: One integer is returned. It is used to indicate error. If the returned value is equal to FFh then an error has occurred in the execution of the command. 'Erdec' can be called to decode the error.

FUNCTION NAME:	int	readl(lenw,addr,rel)
FUNCTION TYPE:		SCSI
PARAMETERS PASSED:		long unsigned int ==> lenw long unsigned int ==> addr unsigned int ==> rel
PARAMETERS RETURNED:		int

PURPOSE: Readl performs the SCSI long read command. It will read data of the specified length from the 4 byte address into the program's data buffer.

REQUIREMENTS: Lenw should be a long unsigned int representing the length of the read to occur in logical sectors. Addr is a long unsigned integer describing the logical sector address for the write on the medium, four bytes of address are possible. Rel specifies the relative addressing mode, it should be one or zero. The bytes describing the address should be modified as necessary if this convention is used,

SPECIAL: The address is specified to be 4 bytes long, as opposed to sread where you only have 3 bytes for address. Length is a word in this function; its value should not exceed the buffer length established before the command occurs. (see read examples.)

EXAMPLES:

```
readl(1,0x208L,0)/*reads one sector from 208h*/

readl(10,1111L,0)/*reads 10 sectors from 1111*/
```

RETURNS: One integer is returned. It is used to indicate error. If the returned value is equal to FFh then an error has occurred in the execution of the command. 'Erdec' can be called to decode the error.

FUNCTION NAME: int recodw()

FUNCTION TYPE: SONY

PARAMETERS PASSED: none

PARAMETERS RETURNED: int

PURPOSE: This command is used to obtain the alternate information from the disk during the disk warning condition.

REQUIREMENTS: Alternate information is 4K in length, a buffer must be established before this command can take place. It will always return 4K of data.

SPECIAL: This command needs a 4K buffer.

EXAMPLE:

```
recodw()  /*reads disk's alternate info(4K)*/
```

RETURNS: One integer is returned. It is used to indicate error. If the returned value is equal to FFh then an error has occurred in the execution of the command. 'Erdec' can be called to decode the error.

FUNCTION NAME: rese()

FUNCTION TYPE: SCSI

PARAMETERS PASSED: none

PARAMETERS RETURNED: int

PURPOSE: Rese is the SCSI command to reset the bus. It is the same as powering down the drive and powering it back up. It can be used to reset the bus if an illegal phase has occurs.

REQUIREMENTS: none.

SPECIAL: Time must be allowed for the reset to occur. Commands directly following a rese call will result in a drive not ready error. It is best to perform a request sense in a loop until the drive is ready.

EXAMPLE:

```
rese();   /* resets the SCSI bus */
```

FUNCTION NAME: int resen(allo)

FUNCTION TYPE: SCSI

PARAMETERS PASSED: unsigned integer ==> allo

PARAMETERS RETURNED: int

PURPOSE: Resen performs the SCSI request sense function. Data is transferred from the target to the initiator of the command regarding the device parameters.

REQUIREMENTS: Allo is an unsigned integer between 0 and 255 specifying the allocation length provided for the information being returned. It should not be greater than the buffer size at the issuing of the command.

SPECIAL: Again, allo should be between 0 and 255. It is usually set to 6. It is important that before this command is executed the buffer be set to at least the size allocated for information to be returned. The data returned with this command is placed in the buffer.(See buffer examples.)

EXAMPLE:

```
resen(0);       /*the short sense command  */

resen(6);       /*the extended data command*/
```

RETURNS: One integer is returned. It is used to indicate error. If the returned value is equal to FFh then an error has occurred in the execution of the command. 'Erdec' can be called to decode the error.

FUNCTION NAME:	int	rodis(allo)
FUNCTION TYPE:		ATG
PARAMETERS PASSED:		unsigned int ==> allo
PARAMETERS RETURNED:		int

PURPOSE: Rodis reads the logical unit's ODI status.

REQUIREMENTS: Allo is the allocated length for returning ODI status information.

SPECIAL: The drive must be put into diagnostic mode before this command can be used. Also it is necessary to remember that allocation length should not exceed the length of the buffer established previously.

EXAMPLE:

```
rodis(10); /*receive 10 bytes of ODI info*/
```

RETURNS: One integer is returned. It is used to indicate error. If the returned value is equal to FFh then an error has occurred in the execution of the command. 'Erdec' can be called to decode the error.

FUNCTION NAME: int rzzo()

FUNCTION TYPE: SCSI

PARAMETERS PASSED: none

PARAMETERS RETURNED: int

PURPOSE: Rzzo is the SCSI command to rezero the drive. The head is repositioned and the medium begins rotation.

REQUIREMENTS: none.

SPECIAL: none.

EXAMPLE:

```
rzzo();   /* rezeros the current target/lun */
```

RETURNS: One integer is returned. It is used to indicate error. If the returned value is equal to FFh then an error has occurred in the execution of the command. 'Erdec' can be called to decode the error

FUNCTION NAME:	int	saiifo()
FUNCTION TYPE:		SONY
PARAMETERS PASSED:		none
PARAMETERS RETURNED:		int

PURPOSE: This reads what is called alternate information by the SONY documentation. Alternate information exists if disk trouble occurred during a read or write.

REQUIREMENTS: Alternate information is 4K in length, a buffer must be established before this command can take place. It will always return 4K of data.

SPECIAL: This command needs a 4K buffer.

EXAMPLE:

```
saiifo()  /*reads disk's alternate info(4K)*/
```

RETURNS: One integer is returned. It is used to indicate error. If the returned value is equal to FFh then an error has occurred in the execution of the command. 'Erdec' can be called to decode the error.

FUNCTION NAME:	int	sarsto (flag)
FUNCTION TYPE:		SCSI
PARAMETERS PASSED:		unsigned integer ==> flag
PARAMETERS RETURNED:		int

PURPOSE: Sarsto performs the start/stop medium function.

REQUIREMENTS: Flag is an unsigned integer and should be only a 1 or a 0. A 1 indicates start medium rotation and a 0 indicates stop medium rotation.

SPECIAL: Flag should be only 1 or 0.

EXAMPLE:

```
sarsto(1);      /*starts medium rotation*/

sarsto(0);      /*stops medium rotation */
```

RETURNS: One integer is returned. It is used to indicate error. If the returned value is equal to FFh then an error has occurred in the execution of the command. 'Erdec' can be called to decode the error.

FUNCTION NAME: int scopy(pllw)

FUNCTION TYPE: SONY

PARAMETERS PASSED: long unsigned int ==> pllw

PARAMETERS RETURNED: int

PURPOSE: Scopy is the data copy command. Data can be copied from disk to disk or onto the same disk by using this command.

REQUIREMENTS: Pllw is a long unsigned integer specifying the list of parameters describing the type of copy to take place. It should by a number from 0 to 256.

SPECIAL: Data describing the type of copy to take place should be present in the buffer before command execution. Construction of such 'copy command data' is discussed in the Sony manual.

EXAMPLE:

```
scopy(12L); /*specifies a param list of 12*/

scopy(256L); /*specifies the longest possible
               parameter list.*/
```

RETURNS: One integer is returned. It is used to indicate error. If the returned value is equal to FFh then an error has occurred in the execution of the command. 'Erdec' can be called to decode the error.

FUNCTION NAME: int scssix(b1,b2,b3,b4,b5,b6)

FUNCTION TYPE: TOOL

PARAMETERS PASSED: unsigned integers ==> b1-b6

PARAMETERS RETURNED: int

PURPOSE: This function uses the six bytes sent to it as the command block.

REQUIREMENTS: All parameters sent must be bytes.

SPECIAL: As the user creates the command block through the parameters sent it is important to be sure of the validity of a command before it is sent. As well, if the command involves the input/output of data or a parameter list the appropriate buffer must be established.

RETURNS: One integer is returned. It is used to indicate error. If the returned value is equal to FFh then an error has occurred in the execution of the command. 'Erdec' can be called to decode the error.

FUNCTION NAME: int scsten(b1,b2,b3,b4,b5,b6,b7,b8,b9,b10)

FUNCTION TYPE: TOOL

PARAMETERS PASSED: unsigned integers ==> b1- b10

PARAMETERS RETURNED: int

PURPOSE: This function uses the ten bytes sent to it as the command block.

REQUIREMENTS: All parameters sent must be bytes.

SPECIAL: As the user creates the command block through the parameters sent it is important to be sure of the validity of a command before it is sent. As well, if the command involves the input/output of data or a parameter list the appropriate buffer must be established.

RETURNS: One integer is returned. It is used to indicate error. If the returned value is equal to FFh then an error has occurred in the execution of the command. 'Erdec' can be called to decode the error.

FUNCTION NAME: int scstwl(b1,b2,b3,b4,b5,b6,b7,b8,b9,b10,b11,b12)

FUNCTION TYPE: TOOL

PARAMETERS PASSED: unsigned integers ==> b1-b12

PARAMETERS RETURNED: int

PURPOSE: This function uses the twelve bytes sent to it as the command block.

REQUIREMENTS: All parameters sent must be bytes.

SPECIAL: As the user creates the command block through the parameters sent it is important to be sure of the validity of a command before it is sent. As well, if the command involves the input/output of data or a parameter list the appropriate buffer must be established.

RETURNS: One integer is returned. It is used to indicate error. If the returned value is equal to FFh then an error has occurred in the execution of the command. 'Erdec' can be called to decode the error.

FUNCTION NAME: int sdisej()

FUNCTION TYPE: SONY

PARAMETERS PASSED: none

PARAMETERS RETURNED: int

PURPOSE: The sdisej command simply ejects the disk from the drive.

REQUIREMENTS: none.

SPECIAL: none.

EXAMPLES:

```
sdisej()  /*ejects the disk from the drive*/
```

RETURNS: One integer is returned. It is used to indicate error. If the returned value is equal to FFh then an error has occurred in the execution of the command. 'Erdec' can be called to decode the error.

FUNCTION NAME:	int	seekbs(addr,rel)
FUNCTION TYPE:		SONY
PARAMETERS PASSED:		long unsigned int ==> addr unsigned int ==> rel
PARAMETERS RETURNED:		int

PURPOSE: Seekbs searches the disk from the specified address until it encounters the first blank sector.

REQUIREMENTS: Addr is a long unsigned integer that specifies four bytes of address. Rel specifies the relative addressing mode when it is equal to 1, and the normal addressing mode when 0.

SPECIAL: There must be a buffer for this command to properly return the address it has found. The buffer should be setup using 'setbufd' before the command is sent.

EXAMPLES:

```
seekbs(1024L,0)  /* searches from the 1k address*/

seekbs(0,1)      /*search from the current addr */
```

RETURNS: One integer is returned. It is used to indicate error. If the returned value is equal to FFh then an error has occurred in the execution of the command. 'Erdec' can be called to decode the error.

FUNCTION NAME:	int	seekl(addr,rel)
FUNCTION TYPE:		ATG
PARAMETERS PASSED:		long unsigned int ==> addr unsigned int ==> rel
PARAMETERS RETURNED:		int

PURPOSE: Seekl moves the laser head of the selected logical unit to the specified address. The address is "pointed to".

REQUIREMENTS: Addr is a long unsigned integer describing the logical sector address. Rel is a 1 or 0 to indicate relative addressing or normal addressing.

SPECIAL: The address is specified to be 4 bytes long; truncation of oversized long integers will occur. The logical sector address should be in the following range: 0 to F423Fh. Because there is no data transferred with this command it is not required to have a buffer for execution.

EXAMPLES:

```
seekl(00L,0)        /*'points' to 0 logical sect*/

seekl(0x207L,0)     /*'points to sector 207      */
```

RETURNS: One integer is returned. It is used to indicate error. If the returned value is equal to FFh then an error has occurred in the execution of the command. 'Erdec' can be called to decode the error.

FUNCTION NAME:	int	seekws(addr,rel)
FUNCTION TYPE:		SONY
PARAMETERS PASSED:		long unsigned int ==> addr unsigned int ==> rel
PARAMETERS RETURNED:		int

PURPOSE: seekws searches the disk from the specified address until it encounters the first written sector.

REQUIREMENTS: Addr is a long unsigned integer that specifies four bytes of address. Rel specifies the relative addressing mode when it is equal to 1, and the normal addressing mode when 0.

SPECIAL: There must be a buffer for this command to properly return the address it has found. The buffer should be setup using 'setbufd' before the command is sent.

EXAMPLES:

```
seekws(1024L,0) /*searches from the 1k address  */

seekws(0L,1)    /*searches from the current addr*/
```

RETURNS: One integer is returned. It is used to indicate error. If the returned value is equal to FFh then an error has occurred in the execution of the command. 'Erdec' can be called to decode the error.

FUNCTION NAME:	setbufd(buffer_length)
FUNCTION TYPE:	TOOL
PARAMETERS PASSED:	unsigned ==> buffer_length
PARAMETERS RETURNED:	none

PURPOSE: Setbufd is a tool function which sets up the data buffer for program use. It must be used before any data transactions can occur. failure to create a buffer will result in a main program crash.

REQUIREMENTS: Setbufd requires one parameter. Buffer_length is the length of the data buffer to be used. The buffer length must be under 10K. 10K is the maximum data buffer size. Buffer_length is expressed in bytes.

SPECIAL: Again, the buffer length can not exceed 10K, and should be expressed in bytes.

EXAMPLE:

```
setbufd(1024);        /* sets a 1K buffer  */
setbufd(10240);       /* sets a 10K buffer */
setbufd(0x800);       /* sets a 2K buffer  */
```

FUNCTION NAME:	int	setmp(mid,irrt,orrt,irwp,orwp)
FUNCTION TYPE:		ATG
PARAMETERS PASSED:		unsigned int ==> mid unsigned int ==> irrt unsigned int ==> orrt unsigned int ==> irwp unsigned int ==> orwp
PARAMETERS RETURNED:		int

PURPOSE: Setmp performs the function allowing the drives medium parameters to be set.

REQUIREMENTS: Mid represents the medium identification, that is, disk style. Irrt, Orrt represent the values of inner and outer radius reading thresholds. Irwp and Orwp represent the inner and outer radius writing power.

SPECIAL: In normal conditions all parameters are set in the drive during the disk spin up. It is not necessary to use this command unless the drive is unable to spin up this information. See the ATG manual for a further discussion.

RETURNS: One integer is returned. It is used to indicate error. If the returned value is equal to FFh then an error has occurred in the execution of the command. 'Erdec' can be called to decode the error.

FUNCTION NAME:	int	smove(addr)
FUNCTION TYPE:		SONY
PARAMETERS PASSED:		long unsigned int ==> addr
PARAMETERS RETURNED:		int

PURPOSE: Smove moves the drive head to the specified sector of a disk.

REQUIREMENTS: Addr is a long unsigned integer equal to a three byte address.

SPECIAL: Remember that only 3 bytes of address are considered.

EXAMPLE:

```
smove(10L); /*moves the head to sector 10*/
```

RETURNS: One integer is returned. It is used to indicate error. If the returned value is equal to FFh then an error has occurred in the execution of the command. 'Erdec' can be called to decode the error.

FUNCTION NAME:	int	srdres(allw)
FUNCTION TYPE:		SONY
PARAMETERS PASSED:		long unsigned int ==> allw
PARAMETERS RETURNED:		int

PURPOSE: Srdres receives diagnostic information formed in the execution of the send diagnostics command.

REQUIREMENTS: Allw specifies the amount of diagnostic data to be returned. Allw is a long unsigned integer of which the lowest two bytes are used.

SPECIAL: Due to the complicated nature of this command it is best to reference the SONY manual.

RETURNS: One integer is returned. It is used to indicate error. If the returned value is equal to FFh then an error has occurred in the execution of the command. 'Erdec' can be called to decode the error.

FUNCTION NAME:	int	sread(length,addr)
FUNCTION TYPE:		SCSI
PARAMETERS PASSED:		unsigned integer ==> length long unsigned int. ==> addr
PARAMETERS RETURNED:		int

PURPOSE: Sread performs the SCSI short read function. Data is transferred from the medium into the specified buffer.

REQUIREMENTS: Length is an unsigned integer between 0 and 255 specifying the length of data to be read. Length has the units of logical sectors. Addr is an long unsigned integer specifying the address of the read data.

SPECIAL: Length should be between 0 and 255 and should not exceed the length of the data buffer set up with the setbufd function, as data is read into this buffer. The address must not exceed three bytes in length, if it does only the 3 least significant bytes are considered as the address.

EXAMPLE:

```
sread(1,204L);     /*reads 1K from addr 204    */

sread(10,0x255L); /*reads 10k from addr 255h */
```

RETURNS: One integer is returned. It is used to indicate error. If the returned value is equal to FFh then an error has occurred in the execution of the command. 'Erdec' can be called to decode the error.

FUNCTION NAME:	int	srele(tp,tpid)
FUNCTION TYPE:		SONY
PARAMETERS PASSED:		unsigned int ==> tp unsigned int ==> tpid
PARAMETERS RETURNED:		int

PURPOSE: Srele releases exclusive control of a drive unit obtained through the reserve function 'srese'.

REQUIREMENTS: If tp=0, no tpid (third party id) is required. If tp=1, then the tpid needs to be specified. In this situation the device the command is sent to is released from exclusive control by the specified party.

SPECIAL: If tp=0 no third party id is necessary.

EXAMPLE:

```
srele(1,2); /*releases the lun from device 2*/

srele(0,0); /*releases the lun from the
              controller*/
```

RETURNS: One integer is returned. It is used to indicate error. If the returned value is equal to FFh then an error has occurred in the execution of the command. 'Erdec' can be called to decode the error.

FUNCTION NAME:	int	srese(tp,tpid)
FUNCTION TYPE:		SONY
PARAMETERS PASSED:		unsigned int ==> tp unsigned int ==> tpid
PARAMETERS RETURNED:		int

PURPOSE: Srese gains exclusive control of a drive unit.

REQUIREMENTS: If Tp=0, no tpid (third party id) is required. If tp=1, then the tpid needs to be specified. In this situation the lun the command is sent to will be reserved by the specified third party.

SPECIAL: If tp=0 no third party id is necessary.

EXAMPLE:

```
srese(1,2);/*reserves the lun for device 2      */

srese(0,0);/*reserves the lun for the controller*/
```

RETURNS: One integer is returned. It is used to indicate error. If the returned value is equal to FFh then an error has occurred in the execution of the command. 'Erdec' can be called to decode the error.

FUNCTION NAME:	int	ssdia(pllw,st,devl,unio)
FUNCTION TYPE:		SONY
PARAMETERS PASSED:		long unsigned int ==> pllw unsigned int ==> st unsigned int ==> devl unsigned int ==> unio
PARAMETERS RETURNED:		int

PURPOSE: Ssdia sends the diagnostic command to execute diagnostic commands with subcommands or a self test.

REQUIREMENTS: Unio is an unsigned integer of 1 or 0. When equal to 1, unio allows writing to occur during diagnostics in the medium test area. If equal to 0 no writing will occur. If devl is set to 1 diagnostics will affect other logical units on the same target. If devl is 0, only the current target/logical unit is affected. St specifies self test when equal to 1. If equal to 0, a level 2 diagnostic is implied. Level two diagnostics require a subcommand. The length of this subcommand should be specified by the long unsigned word pllw, the parameter list length. The parameters that specify the sub command should be in the program's buffer before this function is called.

SPECIAL: Due to the complicated nature of this command it is best to reference the SONY manual.

RETURNS: One integer is returned. It is used to indicate error. If the returned value is equal to FFh then an error has occurred in the execution of the command. 'Erdec' can be called to decode the error.

FUNCTION NAME:	int	sseek(addr)
FUNCTION TYPE:		SCSI
PARAMETERS PASSED:		long unsigned integer ==> addr
PARAMETERS RETURNED:		int

PURPOSE: Sseek performs the SCSI short seek command. This command essentially moves the laser head to the requested sector on the medium. No data transfer occurs.

REQUIREMENTS: Addr is a long unsigned integer. It specifies the address to seek. Only the low three bytes of this variable will be considered as the address. If any bytes higher than three exist, they will be ignored.

SPECIAL: It is important here to remember to properly use the long integer variable type.

EXAMPLE:

```
sseek(0x256L);    /*moves the head to 255h */

sseek(6L);        /*move the head to 6     */
```

RETURNS: One integer is returned. It is used to indicate error. If the returned value is equal to FFh then an error has occurred in the execution of the command. 'Erdec' can be called to decode the error.

FUNCTION NAME: start()

FUNCTION TYPE: SCSI

PARAMETERS PASSED: none

PARAMETERS RETURNED: none

PURPOSE: Start is the drive start function that is developed and added to the library in the lower level program example in the programming portion of the text. Start simply starts medium rotation.

REQUIREMENTS: none

SPECIAL: none

EXAMPLE:

```
start();                /* starts the drive  */
```

FUNCTION NAME:	int	swrit(len,addr)
FUNCTION TYPE:		SCSI
PARAMETERS PASSED:		unsigned int ==>len long unsigned int ==> addr
PARAMETERS RETURNED:		int

PURPOSE: Swrit performs the SCSI short write command. It will write the data supplied in the program buffer to the supplied address. Length is variable controlled.

REQUIREMENTS: Len should be an unsigned integer representing the length of the write in logical sectors. Addr is a long unsigned integer describing the logical sector address for the write on the medium.

SPECIAL: The address is specified to be 3 bytes long. Truncation of oversized long integers will occur. Before this command can occur there must be data to write. Data should be loaded to the buffer before this function is used.

EXAMPLES:

```
swrit(1,0x208L)      /*writes one sector to 208h*/

swrit(10,1111L)      /*writes 10 sectors to 1111 */
```

RETURNS: One integer is returned. It is used to indicate error. If the returned value is equal to FFh then an error has occurred in the execution of the command. 'Erdec' can be called to decode the error.

FUNCTION NAME: int tesur()

FUNCTION TYPE: SCSI

PARAMETERS PASSED: none

PARAMETERS RETURNED: int

PURPOSE: Tesur performs the SCSI test unit ready command.

REQUIREMENTS: none.

SPECIAL: This command can be used in conjunction with the erdec routine to establish the status of a given target.

RETURNS: One integer is returned. It is used to indicate error. If the returned value is equal to FFh then an error has occurred in the execution of the command. 'Erdec' can be called to decode the error.

FUNCTION NAME:	tidlu (targ,lun)
FUNCTION TYPE:	TOOL
PARAMETERS PASSED:	unsigned integer ==> targ unsigned integer ==> lun
PARAMETERS RETURNED:	none

PURPOSE: Tidlu is a TOOL function which initializes the target i.d. and logical unit number in the command block. It can be used to change the target i.d./ logical unit number when addressing a new device is desired.

REQUIREMENTS: Targ and lun represent the target id and the logical unit number respectively. These parameters should be unsigned integers between zero and seven. Failure to conform to these standard limits will result in a random target/lun combination. All following commands will consequently be misdirected.

SPECIAL: Targ and lun must be between 0 and 7.

EXAMPLE:

```
tidlu(0,0);      /* sets the target id and lun to
                    0 and 0. */

tidlu(1,3);      /* sets the target id and lun to
                    1 and 3 respectively. */
```

FUNCTION NAME: userbuf(patlen,fill,start)

FUNCTION TYPE: tool

PARAMETERS PASSED:
unsigned int ==> patlen
unsigned int ==> fill
unsigned int ==> start

PARAMETERS RETURNED: none

PURPOSE: Userbuf is a user interactive subprogram that establishes a pattern then fills the desired portion of the program buffer.

REQUIREMENTS: Patlen is the pattern length represented by an unsigned integer. The user will be prompted for this number of bytes. The fill unsigned integer represents how much of this pattern should be placed in the buffer. Start is also an unsigned integer; it depicts where in the buffer the data will begin.

SPECIAL: The user will be prompted for values of the pattern. The total fill size and the start value should not exceed the upper boundary of the program buffer.

EXAMPLE:

```
userbuf(2,1024,0);  /*2 byte long pattern filling a 1K
                      space in the buffer starting a 0*/

userbuf(12,512,152);/*12 byte long pattern filling a
                      1/2K space in the buffer starting
                      at the 152 byte*/
```

FUNCTION NAME:	int	verif(nos,addr,rel,btch,blkv)
FUNCTION TYPE:		ATG
PARAMETERS PASSED:		long unsigned int ==> nos long unsigned int ==> addr unsigned int ==> rel unsigned int ==> btch unsigned int ==> blkv
PARAMETERS RETURNED:		int

PURPOSE: Verif has several differing functions. It can do a byte by byte check of the specified address. It can also be used to verify that a sector is blank. For a description of how to use the command's different functions refer to the ATG manual.

REQUIREMENTS: Addr is a long unsigned integer describing the logical sector address which is to be verified. Nos is the number of sectors to be checked. Btch specifies that a byte check is to occur; if equal to one a byte by byte check of data occurs. Blkv as a one specifies that a blank sector check is to take place. Rel is a 1 or 0 to indicate relative addressing or normal addressing.

SPECIAL: The address is specified to be 4 bytes long; truncation of oversized long integers will occur. The logical sector address should be in the following range: 0 to F423Fh.

EXAMPLES:

```
verif(1L,6L,0,0,1)  /*verify that 6 is blank   */

verif(1L,6L,0,1,0)  /*verify the data on sect 6*/
```

RETURNS: One integer is returned. It is used to indicate error. If the returned value is equal to FFh then an error has occurred in the execution of the command. 'Erdec' can be called to decode the error.

FUNCTION NAME:	int	writeb(tl,bn)
FUNCTION TYPE:		ATG
PARAMETERS PASSED:		unsigned int ==> tl unsigned int ==> bn
PARAMETERS RETURNED:		int

PURPOSE: Writeb writes data to the specified scsi device drive buffer.

REQUIREMENTS: Tl can be one or zero. When tl=0 the length written is 1024 bytes. When tl=1 the command writes 1115 bytes to the drive buffer. Bn is the buffer number, 0 corresponds to drive buffer 0 and 1 corresponds to drive buffer 1. The command will write the data to the specified drive.

SPECIAL: The drive must be put into diagnostic mode before this command can be used. It is also important to remember that sectors size can be 1115 bytes. The buffer should be able to accommodate this. Data is written from the program buffer to the external device buffer, therefore the program buffer must be loaded with the desired data before this command is executed.

EXAMPLE:

```
writb(1,1); /*writes 1115bytes from the program
              buffer to the drive buffer 1*/

writb(0,0); /*writes 1k from the program buffer to
             the drive buffer 0 */
```

RETURNS: One integer is returned. It is used to indicate error. If the returned value is equal to FFh then an error has occurred in the execution of the command. 'Erdec' can be called to decode the error.

FUNCTION NAME:	int	writl(lenw,addr,rel)
FUNCTION TYPE:		SCSI
PARAMETERS PASSED:		long unsigned int ==> lenw long unsigned int ==> addr unsigned int ==> rel
PARAMETERS RETURNED:		int

PURPOSE: Writl performs the SCSI long write command. It will write the data from the buffer to the supplied address. Length is variable controlled.

REQUIREMENTS: Lenw should be a long unsigned integer representing the length of the write to occur in logical sectors. Addr is a long unsigned integer describing the logical sector address for the write on the medium. Rel specifies the addressing mode (1=relative).

SPECIAL: The address is specified to be 4 bytes long truncation of oversized long integers will occur. Before this command can occur there must be data to write. Data should be loaded to the buffer before this function is used. The length of the write is a long unsigned integer; only the low two bytes of information will be regarded for this parameter.

EXAMPLES:

```
writl(1L,0x208L,0)   /*writes one sector to 208h*/

writl(10L,1111L,0)   /*writes 10 sectors to 1111*/
```

RETURNS: One integer is returned. It is used to indicate error. If the returned value is equal to FFh then an error has occurred in the execution of the command. 'Erdec' can be called to decode the error.

FUNCTION NAME:	int	wrver(lenw,addr,rel,btch)
FUNCTION TYPE:		ATG
PARAMETERS PASSED:		long unsigned int ==> lenw long unsigned int ==> addr unsigned int ==> rel unsigned int ==> btch
PARAMETERS RETURNED:		int

PURPOSE: Wrver performs the ATG write and verify command. It will write the data from the buffer to the supplied address. Length is variable controlled. Mode of data verification can also be controlled.

REQUIREMENTS: Lenw should be a long unsigned int representing the length of the write to occur in logical sectors. Addr is a long unsigned integer describing the logical sector address for the write on the medium. Rel specifies the addressing mode (1=relative). Btch specifies the mode of the command. When btch is 0, a simple medium verification takes place. When 1, a byte by byte data comparison occurs.

SPECIAL: The address is specified to be 4 bytes long; truncation of oversized long integers will occur. Before this command can occur there must be data to write. Data should be loaded to the buffer before this function is used. The length of the write is a long unsigned integer; only the low two bytes of information will be regarded for this parameter.

EXAMPLES:

```
wrver(1L,0x208L,0,0) /*verifies the medium      */

wrver(10L,1111L,0,1) /*verifies the written data*/
```

RETURNS: One integer is returned. It is used to indicate error. If the returned value is equal to FFh then an error has occurred in the execution of the command. 'Erdec' can be called to decode the error.

2.7 FUNCTION SYNTAX LISTINGS

```
#define u unsigned int
#define l long unsigned int
#define v void
#define i int
#define c char[20]
#define b unsigned char

/*************************************************************/
/*   F U N C T I O N   D E F I N I T I O N S                 */
/*************************************************************/

v    ascibuf();                    /* displays buffer as ascii */
v    bufnum(u,u);                  /* displays a buffer section*/
v    byter();                      /* displays byte by byte    */
i    cif(&cmdblk);                 /* command interface funct. */
v    clear();                      /* clears the command block */
v    displbu();                    /* displays the entire buf  */
i    dread(u,l);                   /* diagnostic read          */
i    dwrit(u,l);                   /* diagnostic write         */
i    edsene(u);                    /* eject sense toggle       */
v    erdec();                      /* error decoder            */
v    fillbuf(b,i,i);               /* fill the buffer option   */
i    firbl(l,l,u);                 /* first blank sector search*/
u    hi(l);                        /* returns the high byte    */
i    inqir(u);                     /* the inquiry function     */
u    lo(l);                        /* returns the low byte     */
v    loadbuf(c,u,u);               /* loads the buf from disk  */
v    makebuf(c);                   /* creates a buf cop on disk*/
u    mh(l);                        /* returns the middle high  */
u    ml(l);                        /* returns the middle low   */
i    mosel(u);                     /* mode selection function  */
i    mosen(u);                     /* mode sense function      */
i    prealo(u);                    /* medium removal toggle    */
v    prinbuf();                    /* prints the buffer out    */
i    readb(u,u);                   /* read the drives buffer   */
i    readc(l,u);                   /* read the drive capacity  */
i    readdr(u,l);                  /* read data redundancies   */
i    readid();                     /* read the disk id         */
i    readl(l,l,u);                 /* long read command        */
i    recodw();                     /* recover disk warning     */
i    rese();                       /* reset of the SCSI bus    */
i    resen(u);                     /* request sense            */
i    rodis(u);                     /* read the ODI status      */
i    rzzo();                       /* rezero the drive         */
i    saiifo();                     /* sense alternate info     */
i    sarsto(u);                    /* start/stop toggle        */
i    scopy(l);                     /* the SONY copy command    */
i    scssix(u,u,u,u,u,u);          /* user generated command   */
i    scsten(u,u,u,u,u,u,u,u,u,u);/* user generated command   */
```

```
i    scstwl(u,u,u,u,u,u,u,u,u,u,u,u);/*user generated command*/
i    sdisej();                      /* disk eject command       */
i    seekbs(l,u);                   /* blank sector search      */
i    seekl(l,u);                    /* long seek command        */
i    seekws(l,u);                   /* written sector seek      */
v    setbufd(u);                    /* set up the data buffer   */
i    setmp(u,u,u,u,u);              /* set medium parameters    */
i    smove(l);                      /* the move command         */
i    srdres(l);                     /* receive diagnostics      */
i    sread(u,l);                    /* short read               */
i    srele(u,u);                    /* release command          */
i    srese(u,u);                    /* reserve command          */
i    ssdia(l,u,u,u);                /* send diagnostics         */
i    sseek(l);                      /* short seek               */
v    start();                       /* start medium             */
i    swrit(u,l);                    /* short write              */
i    tesur();                       /* test unit ready          */
v    tidlu(u,u);                    /* set the target/lun nums  */
v    userbuf(u,u,u);                /* user generated buffer    */
i    verif(l,l,u,u,u);              /* verify command           */
i    writeb(u,u);                   /* write to the drive buffer*/
i    writl(l,l,u);                  /* the long write command   */
i    wrver(l,l,u,u);                /* write and verify command */
```

SOURCE CODE

S C S I C O M M A N D S Y S T E M I N S T A L L A T I O N:

Installation of the SCSI COMMAND SYSTEM can be achieved as follows:

FOR TURBO C:

#1] Compile the command interfacing function using the Microsoft Macro Assembler.

C> masm cif.asm

The file 'cif.obj' will be created.

#2] Compile scsilib.c with TURBO C to get the new file 'scsilib.obj'.

#3] Create the library with the TURBO C library manager as follows:

C>TLIB scsilib.lib+scsilib.obj+cif.obj

The system is now ready for use.

TURBO C MAIN PROGRAMS

#1] Main programs for TURBO C should be compiled and executed under a project file like the following, named 'myprj.prj'.

```
progname.c(def.h,scsi.h)
scsilib.lib
```

Where progname is the name of your program. 'Def.h' and 'scsi.h' are command system files and should be included in 'progname.c'.

#2] When compiling the following parameters should be set in TURBO C.

```
PROJECT NAME:        myprj.prj

PRIMARY C FILE:      progname.c

COMPILER:            model: SMALL
                optimization for: SPEED

LINKER:  case sensitive link: OFF
             everything else: ON
```

#3] Use the 'f-9' key option and 'make' the project run.

FOR Microsoft C:

#1] Compile the command interfacing function using the Microsoft Macro Assembler.

C> masm cif.asm

The file 'cif.obj' will be created.

#2] Compile 'scsilib.c' with MSC to get the new file 'scsilib.obj.' as follows:

C> msc scsilib.c

#3] Create the library with the MSC library manager as follows:

C>LIB scsilib.lib+scsilib.obj+cif.obj

The system is now ready for use.

MSC MAIN PROGRAMS

#1] Main programs for MSC should include the files 'def.h' and 'scsi.h'. Take for instance 'program.c':

C>MSC program.c

will yield the object file 'program.obj'. 'Program.obj' can be linked as follows:

C>link program.obj

At the linker's 'lib' prompt the following should be answered:

Libraries [.LIB]: scsilib.lib

This should resolve all external calls. The resulting file can be run.

```
/*********************************************************************/
/*                                                                   */
/*     NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY                */
/*     -------- --------- -- --------- --- ----------               */
/*            scsi command software for optical                      */
/*          disk controlers meeting this standard                    */
/*          [for use with the M.D.I. scsi-1 board ]                  */
/*                                                                   */
/*       S C S I    F U N C T I O N     L I B R A R Y                */
/* developed by:  J.GORCZYCA and EDUARDO SANCHEZ VILLAGRAN           */
/*                                                                   */
/*********************************************************************/


/*********************************************************************/
/* D E F I N I T I O N S   A N D   S T R U C T U R E S               */
/*********************************************************************/

#include <stdio.h>
#include <dos.h>

#define byte unsigned char
#define word unsigned int

typedef struct{          /*defines the command block structure     */
     byte cmd;           /*the driver command (00 or 60)           */
     byte tidlun;        /*target id and logical unit number       */
     byte cdb[12];       /*a 12 byte command descriptor block      */
     byte scsirc;        /*the driver return code                  */
     byte targrc;        /*the target return code                  */
     word bufseg;        /*the buffer segment                      */
     word bufoff;        /*the buffer offset                       */
     word buflen;        /*the buffer length                       */
     word senseg;        /*the sense segment                       */
     word senoff;        /*the sense offset                        */
     word senlen;        /*the sense length                        */
               )CMDBLK;  /*end of the definition                   */
     byte tar;           /*the target id                           */
     byte lun;           /*the logical unit number                 */
     CMDBLK cmdblk;      /*define cmdblk as CMDBLK type            */


struct{
     byte by[10240];
     }buffer;

struct SREGS segregs;

/*********************************************************************/
/* S E T   U P   D A T A   B U F F E R                               */
/*********************************************************************/

setbufd(len)
int len;
{
     segread(&segregs);
     cmdblk.bufseg=segregs.ds;
     cmdblk.bufoff= (unsigned) &buffer;
     cmdblk.buflen=len;

}


/*********************************************************************/
/* T A R G E T   I D // L U N   R E S E T   S U B R O U T I N E      */
/*********************************************************************/

tidlu(tar,lun)
byte tar,lun;
{
tar=tar<<5;
cmdblk.tidlun=tar | lun;
cmdblk.tidlun=cmdblk.tidlun & 0xe7;
tar=tar>>5;
cmdblk.cdb[1]=lun<<5;
return;
}


/*********************************************************************/
/* B U S   R E S E T   S U B R O U T I N E                           */
/*********************************************************************/

rese()
{
cmdblk.cmd=0x60;
return(cif(&cmdblk));
}
```

```
/*********************************************************************/
/* R E Z E R O   S U B R O U T I N E                                 */
/*********************************************************************/

rzzo()
{
clear();
cmdblk.cdb[0]=1;
cmdblk.cdb[1]=cmdblk.cdb[1] & 0xe0;
return(cif(&cmdblk));
}
```

```
/*********************************************************************/
/* P R E V E N T / A L L O W   M E D I U M   R E M O V A L           */
/*********************************************************************/

prealo(flag)
byte flag;
{
clear();
cmdblk.cdb[0]=0x1e;
cmdblk.cdb[1]=cmdblk.cdb[1] & 0xe0;
cmdblk.cdb[4]=flag;
return(cif(&cmdblk));
}
```

```
/*********************************************************************/
/* S T A R T/ S T O P   F U N C T I O N                              */
/*********************************************************************/

sarsto(flag)
byte flag;
{
clear();
cmdblk.cdb[0]=0x1b;
cmdblk.cdb[1]=cmdblk.cdb[1] & 0xe0;
cmdblk.cdb[4]=flag;
return(cif(&cmdblk));
}
```

```
/*********************************************************************/
/* I N Q U I R Y   F U N C T I O N                                   */
/*********************************************************************/

inqir(allo)
byte allo;
{
clear();
cmdblk.cdb[0]=0x12;
cmdblk.cdb[1]=cmdblk.cdb[1] & 0xe0;
cmdblk.cdb[4]=allo;
return(cif(&cmdblk));
}
```

```
/*********************************************************************/
/* M O D E   S E L E C T   F U N C T I O N                           */
/*********************************************************************/

mosel(pll)
byte pll;
{
clear();
cmdblk.cdb[0]=0x15;
cmdblk.cdb[1]=cmdblk.cdb[1] & 0xe0;
cmdblk.cdb[4]=pll;
return(cif(&cmdblk));
}

/*********************************************************************/
/* M O D E   S E N S E    F U N C T I O N                            */
/*********************************************************************/

mosen(allo)
byte allo;
{
clear();
cmdblk.cdb[0]=0x1a;
cmdblk.cdb[1]=cmdblk.cdb[1] & 0xe0;
cmdblk.cdb[4]=allo;
return(cif(&cmdblk));
}
```

```
/*****************************************************************/
/* S H O R T   R E A D    F U N C T I O N                        */
/*****************************************************************/

sread(len,addr)
long addr;
byte len;
{
byte ah,am,al;
clear();
ah=mh(addr);
am=ml(addr);
al=lo(addr);
cmdblk.cdb[0]=0x08;
cmdblk.cdb[1]=(cmdblk.cdb[1] & 0xe0) | ah;
cmdblk.cdb[2]=am;
cmdblk.cdb[3]=al;
cmdblk.cdb[4]=len;
return(cif(&cmdblk));
}

/*****************************************************************/
/* R E Q U E S T   S E N S E   F U N C T I O N                   */
/*****************************************************************/

resen(allo)
byte allo;
{
clear();
cmdblk.cdb[0]=0x3;
cmdblk.cdb[1]=cmdblk.cdb[1] & 0xe0;
cmdblk.cdb[4]=allo;
return(cif(&cmdblk));
}


/*****************************************************************/
/* S H O R T   S E E K    F U N C T I O N                        */
/*****************************************************************/

sseek(addr)
long addr;
{
byte am,al,ah;
clear();
ah=mh(addr);
am=ml(addr);
al=lo(addr);
cmdblk.cdb[0]=0x0B;
cmdblk.cdb[1]=(cmdblk.cdb[1] & 0xe0) | ah;
cmdblk.cdb[2]=am;
cmdblk.cdb[3]=al;
return(cif(&cmdblk));
}


/*****************************************************************/
/* T E S T   U N I T   R E A D Y    F U N C T I O N              */
/*****************************************************************/

tesur()
{
clear();
cmdblk.cdb[0]=0;
cmdblk.cdb[1]=cmdblk.cdb[1] & 0xe0;
return(cif(&cmdblk));
}

/*****************************************************************/
/* S H O R T   W R I T E   F U N C T I O N                       */
/*****************************************************************/

swrit(len,addr)
long addr;
byte len;
{
byte ah,am,al;
clear();
ah=mh(addr);
am=ml(addr);
al=lo(addr);
cmdblk.cdb[0]=0x0A;
cmdblk.cdb[1]=(cmdblk.cdb[1] & 0xe0) | ah;
cmdblk.cdb[2]=am;
cmdblk.cdb[3]=al;
cmdblk.cdb[4]=len;
return(cif(&cmdblk));
}
```

```
/*********************************************************************/
/* L O N G    R E A D    F U N C T I O N                    */
/*********************************************************************/

readl(lenw,addr,rel)
long addr,lenw;
byte rel;
{
byte llen,ulen,ah,amh,aml,al;
clear();
llen=lo(lenw);
ulen=ml(lenw);
ah=hi(addr);
amh=mh(addr);
aml=ml(addr);
al=lo(addr);
cmdblk.cdb[0]=0x28;
cmdblk.cdb[1]=(cmdblk.cdb[1] & 0xe0) | rel;
cmdblk.cdb[2]=ah;
cmdblk.cdb[3]=amh;
cmdblk.cdb[4]=aml;
cmdblk.cdb[5]=al;
cmdblk.cdb[7]=ulen;
cmdblk.cdb[8]=llen;
return(cif(&cmdblk));
}


/*********************************************************************/
/* R E A D   C A P A C I T Y   F U N C T I O N              */
/*********************************************************************/

readc(addr,rel)
long addr;
byte rel;
{
byte ah,amh,aml,al;
clear();
ah=hi(addr);
amh=mh(addr);
aml=ml(addr);
al=lo(addr);
cmdblk.cdb[0]=0x25;
cmdblk.cdb[1]=(cmdblk.cdb[1] & 0xe0) | rel;
cmdblk.cdb[2]=ah;
cmdblk.cdb[3]=amh;
cmdblk.cdb[4]=aml;
cmdblk.cdb[5]=al;
return(cif(&cmdblk));
}



/*********************************************************************/
/* L O N G   S E E K   F U N C T I O N                     */
/*********************************************************************/

seekl(addr,rel)
long addr;
byte  rel;
{
byte ah,amh,aml,al;
clear();
ah=hi(addr);
amh=mh(addr);
aml=ml(addr);
al=lo(addr);
cmdblk.cdb[0]=0x2B;
cmdblk.cdb[1]=(cmdblk.cdb[1] & 0xe0) | rel;
cmdblk.cdb[2]=ah;
cmdblk.cdb[3]=amh;
cmdblk.cdb[4]=aml;
cmdblk.cdb[5]=al;
return(cif(&cmdblk));
}


/*********************************************************************/
/* V E R I F Y   F U N C T I O N                           */
/*********************************************************************/

verif(nos,addr,rel,byc,blv)
long addr,nos;
byte rel,blv,byc;
{
byte nsh,nsl,ah,amh,aml,al;
clear();
nsl=lo(nos);
nsh=ml(nos);
ah=hi(addr);
amh=mh(addr);
aml=ml(addr);
```

```
al=lo(addr);
blv=blv<<2;
byc=byc<<1;
cmdblk.cdb[0]=0x2f;
cmdblk.cdb[1]=(cmdblk.cdb[1] & 0xe0) | rel | blv | byc;
cmdblk.cdb[2]=ah;
cmdblk.cdb[3]=amh;
cmdblk.cdb[4]=aml;
cmdblk.cdb[5]=al;
cmdblk.cdb[7]=nsh;
cmdblk.cdb[8]=nsl;
return(cif(&cmdblk));
}


/*********************************************************************/
/* L O N G    W R I T E  F U N C T I O N                              */
/*********************************************************************/

writl(lenw,addr,rel)
long addr,lenw;
byte rel;
{
byte llen,ulen,ah,amh,aml,al;
clear();
llen=lo(lenw);
ulen=ml(lenw);
ah=hi(addr);
amh=mh(addr);
aml=ml(addr);
al=lo(addr);
cmdblk.cdb[0]=0x2a;
cmdblk.cdb[1]=(cmdblk.cdb[1] & 0xe0) | rel;
cmdblk.cdb[2]=ah;
cmdblk.cdb[3]=amh;
cmdblk.cdb[4]=aml;
cmdblk.cdb[5]=al;
cmdblk.cdb[7]=ulen;
cmdblk.cdb[8]=llen;
return(cif(&cmdblk));
}



/*********************************************************************/
/* W R I T E  V E R I F Y  F U N C T I O N                           */
```

```
/*********************************************************************/

wrver(lenw,addr,rel,byc)
long addr,lenw;
byte rel,byc;
{
byte llen,ulen,ah,amh,aml,al,lah;
clear();
llen=lo(lenw);
ulen=ml(lenw);
ah=hi(addr);
amh=mh(addr);
aml=ml(addr);
al=lo(addr);
byc=byc<<1;
cmdblk.cdb[0]=0x2e;
cmdblk.cdb[1]=(cmdblk.cdb[1] & 0xe0) | rel | byc;
cmdblk.cdb[2]=ah;
cmdblk.cdb[3]=amh;
cmdblk.cdb[4]=aml;
cmdblk.cdb[5]=al;
cmdblk.cdb[7]=ulen;
cmdblk.cdb[8]=llen;
return(cif(&cmdblk));
}




/*********************************************************************/
/* F I R S T   B L A N K    F U N C T I O N                          */
/*********************************************************************/

firbl(nos,addr,rel)
long addr,nos;
byte rel;
{
byte nsh,nsl,ah,amh,aml,al,lah;
clear();
nsl=lo(nos);
nsh=ml(nos);
ah=hi(addr);
amh=mh(addr);
aml=ml(addr);
al=lo(addr);
cmdblk.cdb[0]=0xD0;
```

```
cmdblk.cdb[1]=(cmdblk.cdb[1] & 0xe0) | rel;
cmdblk.cdb[2]=ah;
cmdblk.cdb[3]=amh;
cmdblk.cdb[4]=aml;
cmdblk.cdb[5]=al;
cmdblk.cdb[7]=nsh;
cmdblk.cdb[8]=nsl;
return(cif(&cmdblk));
}

/*********************************************************************/
/* S E T   M E D I U M  P A R A M E T E R S                          */
/*********************************************************************/

setmp(mid,irrt,orrt,irwp,orwp)
byte mid,irrt,orrt,irwp,orwp;
{
clear();
cmdblk.cdb[0]=0xDD;
cmdblk.cdb[1]=cmdblk.cdb[1] & 0xe0;
cmdblk.cdb[4]=mid;
cmdblk.cdb[5]=irrt;
cmdblk.cdb[6]=orrt;
cmdblk.cdb[7]=irwp;
cmdblk.cdb[8]=orwp;
return(cif(&cmdblk));
}



/*********************************************************************/
/* D I A G N O S T I C  R E A D  F U N C T I O N                     */
/*********************************************************************/

dread(len,addr)
long addr;
byte len;
{
byte ah,am,al;
clear();
ah=mh(addr);
am=ml(addr);
al=lo(addr);
cmdblk.cdb[0]=0xe8;
cmdblk.cdb[1]=(cmdblk.cdb[1] & 0xe0)  |  ah;
cmdblk.cdb[2]=am;
cmdblk.cdb[3]=al;
cmdblk.cdb[4]=len;
return(cif(&cmdblk));
}



/*********************************************************************/
/* D I A G N O S T I C  W R I T E  F U N C T I O N                   */
/*********************************************************************/

dwrit(len,addr)
long addr;
byte len;
{
byte ah,am,al;
clear();
ah=mh(addr);
am=ml(addr);
al=lo(addr);
cmdblk.cdb[0]=0x28;
cmdblk.cdb[1]=(cmdblk.cdb[1] & 0xe0) | ah;
cmdblk.cdb[2]=am;
cmdblk.cdb[3]=al;
cmdblk.cdb[4]=len;
return(cif(&cmdblk));
}

/*********************************************************************/
/* R E A D  B U F F E R  F U N C T I O N                             */
/*********************************************************************/

readb(tl,bn)
byte tl,bn;
{
clear();
cmdblk.cdb[0]=0xf8;
cmdblk.cdb[1]=cmdblk.cdb[1] & 0xe0;
cmdblk.cdb[3]=tl;
cmdblk.cdb[4]=bn;
return(cif(&cmdblk));
}

/*********************************************************************/
/* W R I T E  B U F F E R  F U N C T I O N                           */
/*********************************************************************/
```

```
writeb(tl,bn)
byte tl,bn;
{
clear();
cmdblk.cdb[0]=0xfa;
cmdblk.cdb[1]=cmdblk.cdb[1] & 0xe0;
cmdblk.cdb[3]=tl;
cmdblk.cdb[4]=bn;
return(cif(&cmdblk));
}

/*********************************************************************/
/* R E A D  D A T A  R E D U N D E N C Y  F U N C T I O N       */
/*********************************************************************/

readdr(len,addr)
long addr;
byte len;
{
byte ah,am,al;
clear();
ah=mh(addr);
am=ml(addr);
al=lo(addr);
cmdblk.cdb[0]=0xFE;
cmdblk.cdb[1]=(cmdblk.cdb[1] & 0xe0) | ah;
cmdblk.cdb[2]=am;
cmdblk.cdb[3]=al;
cmdblk.cdb[4]=len;
return(cif(&cmdblk));
}

/*********************************************************************/
/* R E A D  O D I  S T A T U S  F U N C T I O N                */
/*********************************************************************/

rodis(allo)
byte allo;
{
clear();
cmdblk.cdb[0]=0xF6;
cmdblk.cdb[1]=cmdblk.cdb[1] & 0xe0;
cmdblk.cdb[4]=allo;
return(cif(&cmdblk));
}
```

```
/*********************************************************************/
/* S O N Y   M O V E   F U N C T I O N                        */
/*********************************************************************/

smove(addr)
long addr;
{
byte ah,am,al;
clear();
ah=mh(addr);
am=ml(addr);
al=lo(addr);
cmdblk.cdb[0]=0x0C;
cmdblk.cdb[1]=(cmdblk.cdb[1] & 0xe0) | ah;;
cmdblk.cdb[2]=am;
cmdblk.cdb[3]=al;
return(cif(&cmdblk));
}

/*********************************************************************/
/* S O N Y   R E S E R V E   F U N C T I O N                  */
/*********************************************************************/

srese(tp,tpid)
byte tp,tpid;
{
clear();
tpid=tpid<<1;
tp=tp<<4;
cmdblk.cdb[0]=0x16;
cmdblk.cdb[1]=((cmdblk.cdb[1] & 0xe0) | tp | tpid) & 0xfe;
return(cif(&cmdblk));
}


/*********************************************************************/
/* S O N Y   R E L E A S E   F U N C T I O N                  */
/*********************************************************************/

srele(tp,tpid)
byte tp,tpid;
{
clear();
```

```
tpid=tpid<<1;
tp=tp<<4;
cmdblk.cdb[0]=0x17;
cmdblk.cdb[1]=((cmdblk.cdb[1] & 0xe0) | tp | tpid) & 0xfe;
return(cif(&cmdblk));
}


/*********************************************************************/
/** S E N D   D I A G N O S T I C S   F U N C T I O N           */
/*********************************************************************/

ssdia(pllw,st,devl,unio)
byte devl,unio,st;
long pllw;
{
byte ph,pl;
clear();
st=st<<2;
devl=devl<<2;
ph=ml(pllw);
pl=lo(pllw);
cmdblk.cdb[0]=0x1D;
cmdblk.cdb[1]=(cmdblk.cdb[1] & 0xe0) | st | devl | unio;
cmdblk.cdb[3]=ph;
cmdblk.cdb[4]=pl;
return(cif(&cmdblk));
}



/*********************************************************************/
/* R E C I E V E   D I A G O N O S T I C   R E S U L T         */
/*********************************************************************/

srdres(allw)
long allw;
{
byte allh,allo;
clear();
allh=ml(allw);
allo=lo(allw);
cmdblk.cdb[0]=0x1C;
cmdblk.cdb[1]=cmdblk.cdb[1] & 0xe0;
cmdblk.cdb[3]=allh;
cmdblk.cdb[4]=allo;
return(cif(&cmdblk));
}


/*********************************************************************/
/* S O N Y   C O P Y   F U N C T I O N                         */
/*********************************************************************/

scopy(pllw)
long pllw;
{
byte ph,pm,pl;
clear();
ph=mh(pllw);
pm=ml(pllw);
pl=lo(pllw);
cmdblk.cdb[0]=0x18;
cmdblk.cdb[1]=cmdblk.cdb[1] & 0xe0;
cmdblk.cdb[2]=ph;
cmdblk.cdb[3]=pm;
cmdblk.cdb[4]=pl;
return(cif(&cmdblk));
}


/*********************************************************************/
/* B L A N K   S E C T O R   S E A R C H   for sony            */
/*********************************************************************/

seekbs(addr,rel)
long addr;
byte rel;
{
byte ah,amh,aml,al;
clear();
ah=hi(addr);
amh=mh(addr);
aml=ml(addr);
al=lo(addr);
cmdblk.cdb[0]=0x2c;
cmdblk.cdb[1]=(cmdblk.cdb[1] & 0xe0) | rel;
cmdblk.cdb[2]=ah;
cmdblk.cdb[3]=amh;
cmdblk.cdb[4]=aml;
cmdblk.cdb[5]=al;
return(cif(&cmdblk));
```

```
}


/*********************************************************************/
/* W R I T T E N  S E C T O R  S E A R C H for sony                   */
/*********************************************************************/

seekws(addr,rel)
long addr;
byte rel;
{
byte ah,amh,aml,al;
clear();
ah=hi(addr);
amh=mh(addr);
aml=ml(addr);
al=lo(addr);
cmdblk.cdb[0]=0x2D;
cmdblk.cdb[1]=(cmdblk.cdb[1] & 0xe0) | rel;
cmdblk.cdb[2]=ah;
cmdblk.cdb[3]=amh;
cmdblk.cdb[4]=aml;
cmdblk.cdb[5]=al;
return(cif(&cmdblk));
}

/*********************************************************************/
/* D I S K  E J E C T  F U N C T I O N                                */
/*********************************************************************/

sdisej()
{
clear();
cmdblk.cdb[0]=0xC0;
cmdblk.cdb[1]=cmdblk.cdb[1] & 0xe0;
return(cif(&cmdblk));
}



/*********************************************************************/
/* E N A B L E / D I S A B L E  R E Q U E S T  S E N S E             */
/*********************************************************************/

edsene(flag)
byte flag;
{
clear();
cmdblk.cdb[0]=0xC1;
cmdblk.cdb[1]=cmdblk.cdb[1] & 0xe0;
cmdblk.cdb[4]=flag;
return(cif(&cmdblk));
}


/*********************************************************************/
/* R E A D  D I S K  I D  F U N C T I O N                             */
/*********************************************************************/

readid()
{
clear();
cmdblk.cdb[0]=0xC2;
cmdblk.cdb[1]=cmdblk.cdb[1] & 0xe0;
return(cif(&cmdblk));
}



/*********************************************************************/
/* S E N S E  A L T E R N A T E  I N F O R M A T I O N  sony         */
/*********************************************************************/

saiifo()
{
clear();
cmdblk.cdb[0]=0xC3;
cmdblk.cdb[1]=cmdblk.cdb[1] & 0xe0;
return(cif(&cmdblk));
}



/*********************************************************************/
/* R E C O V E R  D I S K  W A R N I N G  F U N C T I O N            */
/*********************************************************************/

recodw()
{
clear();
cmdblk.cdb[0]=0xC4;
cmdblk.cdb[1]=cmdblk.cdb[1] & 0xe0;
```

```
return(cif(&cmdblk));
}


/*****************************************************************/
/* T Y P E   C O N V E R S I O N   S U B R O U T I N E S     */
/*****************************************************************/
/*   RETURNS THE HI BYTE OF A LONG INTEGER                       */

hi(forby)
unsigned long forby;
{
byte ahv;
forby=forby>>24;
ahv=forby & 0xffL;
return(ahv);
}


/*   RETURNS THE HI MIDDLE BYTE OF A LONG INT                    */

mh(forby)
unsigned long forby;
{
byte mhv;
forby=forby>>16;
mhv=forby & 0xffL;
return(mhv);
}


/*   RETURNS THE MIDDLE LOW BYTE OF A LONG INT                   */

ml(forby)
unsigned long forby;
{
byte mlv;
forby=forby>>8;
mlv=forby & 0xff;
return(mlv);
}



lo(forby)
/*   RETURNS THE HI BYTE OF A LONG INTEGER                       */
unsigned long forby;
{
byte lo;
lo=forby & 0xff;
return(lo);
}

/*****************************************************************/
/* C L E A R   C O M M A N D  B L O C K  F U N C T I O N     */
/*****************************************************************/

clear()
{
cmdblk.cmd=0;
cmdblk.cdb[0]=0;
cmdblk.cdb[2]=0;
cmdblk.cdb[3]=0;
cmdblk.cdb[4]=0;
cmdblk.cdb[5]=0;
cmdblk.cdb[6]=0;
cmdblk.cdb[7]=0;
cmdblk.cdb[8]=0;
cmdblk.cdb[9]=0;
cmdblk.cdb[10]=0;
cmdblk.cdb[11]=0;
cmdblk.cdb[12]=0;
return;
}
/*****************************************************************/
/* 0 6     U S E R  P R O G R A M E D  F U N C T I O N       */
/*****************************************************************/

scssix(by0,by1,by2,by3,by4,by5)
byte by0,by1,by2,by3,by4,by5;
{
clear();
cmdblk.cdb[0]=by0;
cmdblk.cdb[1]=by1;
cmdblk.cdb[2]=by2;
cmdblk.cdb[3]=by3;
cmdblk.cdb[4]=by4;
cmdblk.cdb[5]=by5;
return(cif(&cmdblk));
}

/*****************************************************************/
```

```
/*  10     U S E R   P R O G R A M E D   F U N C T I O N          */
/*************************************************************/

scsten(by0,by1,by2,by3,by4,by5,by6,by7,by8,by9)
byte by0,by1,by2,by3,by4,by5,by6,by7,by8,by9;
{
clear();
cmdblk.cdb[0]=by0;
cmdblk.cdb[1]=by1;
cmdblk.cdb[2]=by2;
cmdblk.cdb[3]=by3;
cmdblk.cdb[4]=by4;
cmdblk.cdb[5]=by5;
cmdblk.cdb[6]=by6;
cmdblk.cdb[7]=by7;
cmdblk.cdb[8]=by8;
cmdblk.cdb[9]=by9;
return(cif(&cmdblk));
}

/*************************************************************/
/*  12     U S E R   P R O G R A M E D   F U N C T I O N          */
/*************************************************************/

scstwl(by0,by1,by2,by3,by4,by5,by6,by7,by8,by9,by10,by11)
byte by0,by1,by2,by3,by4,by5,by6,by7,by8,by9,by10,by11;
{
clear();
cmdblk.cdb[0]=by0;
cmdblk.cdb[1]=by1;
cmdblk.cdb[2]=by2;
cmdblk.cdb[3]=by3;
cmdblk.cdb[4]=by4;
cmdblk.cdb[5]=by5;
cmdblk.cdb[6]=by6;
cmdblk.cdb[7]=by7;
cmdblk.cdb[8]=by8;
cmdblk.cdb[9]=by9;
cmdblk.cdb[10]=by10;
cmdblk.cdb[11]=by11;
return(cif(&cmdblk));
}

/*************************************************************/
/* E R R O R   D E C O D E R   S U B P R O G R A M            */
/*************************************************************/

erdec()
{

resen(0);

switch(buffer.by[0] & 0x7f){

     case 0:
     printf("CLASS   0   ERROR   0\n");
     printf(" ATG:  no error detected\n");
     printf("SONY:  no error detected\n");
     break;

     case 1:
     printf("CLASS   0   ERROR   1\n");
     printf(" ATG:  no error detected\n");
     printf("SONY:  invalid command \n ");
     break;

     case 2:
     printf("CLASS   0   ERROR   2\n");
     printf(" ATG:  seek error occured\n");
     printf("SONY:  recovered error \n ");
     break;

     case 3:
     printf("CLASS   0   ERROR   3\n");
     printf(" ATG:  cartridge replaced since previous command\n");
     printf("SONY:  illegal request \n ");
     break;

     case 4:
     printf("CLASS   0   ERROR   4\n");
     printf(" ATG:  drive not ready\n  ");
     printf("SONY:  no error detected\n");
     break;

     case 5:
     printf("CLASS   0   ERROR   5\n");
     printf(" ATG:  controller busy \n ");
     printf("SONY:  no error detected\n");
     break;

     case 6:
     printf("CLASS   0   ERROR   6\n");
```

```
printf(" ATG:  reset has occured since previous command\n");
printf("SONY:  unit attention condition\n");
break;

case 7:
printf("CLASS   0   ERROR   7\n");
printf(" ATG:  ACK signal not present after REQUEST\n");
printf("SONY:  parity error detected\n");
break;

case 8:
printf("CLASS   0   ERROR   8\n");
printf(" ATG:  fatal error in global operation\n");
printf("SONY:  message rejected error detected\n");
break;

case 10:
printf("CLASS   0   ERROR   A\n");
printf(" ATG:  invalid set up of parameters\n");
printf("SONY:  copy aborted \n");
break;

case 11:
printf("CLASS   0   ERROR   B\n");
printf(" ATG:  reset and cartridge replacement have occured\n");
printf("SONY:  initiator has detected error, command aborted\n");
break;

case 12:
printf("CLASS   0   ERROR   C\n");
printf(" ATG:  cartridge replacement and invalid auto-set up error\n");
printf("SONY:  select-reselect has failed, command aborted \n");
break;

case 13:
printf("CLASS   0   ERROR   D\n");
printf(" ATG:  reset to an invalid set of parameters has occured\n");
printf("SONY:  no error has occured\n");
break;

case 14:
printf("CLASS   0   ERROR   E\n");
printf(" ATG:  invalid set up of parameters\n");
printf("SONY:  no error has occured \n");
break;

    default:
    err1(buffer.by[0] & 0x7f);
    break;
    }
return;
}
    err1(ecc)
    int ecc;
    {
    switch(ecc){

    case 16:
    printf("CLASS   1   ERROR   0\n");
    printf(" ATG:  uncorrectable error on pre-grooved address\n");
    printf("SONY:  ECC error has occured\n");
    break;

    case 0x11:
    printf("CLASS   1   ERROR   1\n");
    printf(" ATG:  uncorrectable error in data field, EDAC or CRC error\n");
    printf("SONY:  tiem out error       \n");
    break;

    case 0x12:
    printf("CLASS   1   ERROR   2\n");
    printf(" ATG:  unreadable sync marks, acess has failed \n");
    printf("SONY:  controller error has occured\n");
    break;

    case 0x13:
    printf("CLASS   1   ERROR   3\n");
    printf(" ATG:  non blank sector detected durring command\n");
    printf("SONY:  no error has occured \n");
    break;

    case 0x14:
    printf("CLASS   1   ERROR   4\n");
    printf(" ATG:  mismatch durring VERIFY operation\n");
    printf("SONY:  no error has occured\n");
    break;

    case 0x15:
    printf("CLASS   1   ERROR   5\n");
    printf(" ATG:  bad sectors on the disk\n");
    printf("SONY:  unmatched ROM error  \n");
    break;
```

```
    case 0x16:
    printf("CLASS   1   ERROR   6\n");
    printf(" ATG:  uncorrectable error on pre-grooved address\n");
    printf("SONY:  ECC error has occured\n");
    break;

    case 0x17:
    printf("CLASS   1   ERROR   7\n");
    printf(" ATG:  errors in data exceed EDAC threshold      \n");
    printf("SONY:  no error has occured \n");
    break;

    case 0x18:
    printf("CLASS   1   ERROR   8\n");
    printf(" ATG:  delayed error reported  \n");
    printf("SONY:  no error has occured \n");
    break;

    case 0x19:
    printf("CLASS   1   ERROR   9\n");
    printf(" ATG:  data error in header bytes of EDAC pause \n");
    printf("SONY:  no error has occured \n");
    break;

    case 0x1a:
    printf("CLASS   1   ERROR   A\n");
    printf(" ATG:  sector can not be accessed \n");
    printf("SONY:  no error has occured \n");
    break;

    case 0x1b:
    printf("CLASS   1   ERROR   B\n");
    printf(" ATG:  logical sector address does not match requested\n");
    printf("SONY:  no error has occured \n");
    break;

    default:
    err2(ecc);
    break;
    }
return;
}


    err2(ecc)
int ecc;
{
switch(ecc){

case 0x20:
printf("CLASS   2   ERROR   0\n");
printf(" ATG:  command is not valid\n");
printf("SONY:  command not terminated, interface timeout\n");
break;

case 0x21:
printf("CLASS   2   ERROR   1\n");
printf(" ATG:  sector is out of medium range\n");
printf("SONY:  drive interface parity error\n");
break;

case 0x22:
printf("CLASS   2   ERROR   2\n");
printf(" ATG:  command recieved by target is not valid \n");
printf("SONY:  disk loading difficulty\n");
break;

case 0x23:
printf("CLASS   2   ERROR   3\n");
printf(" ATG:  requested sector out of range \n");
printf("SONY:  focusing servos have failed\n");
break;

case 0x24:
printf("CLASS   2   ERROR   4\n");
printf(" ATG:  access to group-7 commands violation \n");
printf("SONY:  tracking servos have failed\n");
break;

case 0x25:
printf("CLASS   2   ERROR   5\n");
printf(" ATG:  command requires that EDAC be present\n");
printf("SONY:  spindle servo failure\n");
break;

case 0x26:
printf("CLASS   2   ERROR   6\n");
printf(" ATG:  protocol error on the bus \n");
printf("SONY:  slide trouble in the servo system\n");
break;
```

```
        case 0x27:
        printf("CLASS   2   ERROR   7\n");
        printf(" ATG:  attempt to write on prewritten medium \n");
        printf("SONY:  skew trouble in hardware \n");
        break;

        case 0x28:
        printf("CLASS   2   ERROR   8\n");
        printf(" ATG:  invalid mode select combination\n");
        printf("SONY:  laser head has left the medium\n");
        break;

        case 0x29:
        printf("CLASS   2   ERROR   9\n");
        printf(" ATG:  command not implemented\n");
        printf("SONY:  write modulation difficulty\n");
        break;

        case 0x2a:
        printf("CLASS   2   ERROR   A\n");
        printf(" ATG:  record mismatch\n");
        printf("SONY:  laser power is too low to execute command\n");
        break;

        case 0x2b:
        printf("CLASS   2   ERROR   B\n");
        printf(" ATG:  sense data is lost\n");
        printf("SONY:  laser power is too high to execute command\n");
        break;

        case 0x2c:
        printf("CLASS   2   ERROR   C\n");
        printf(" ATG:  logical unit reserved\n");
        printf("SONY:  no error has occured \n");
        break;

        default:
        err3(ecc);
        break;
        }
return;
}
        err3(ecc)
        int ecc;
        {
        switch(ecc){
        case 0x30:
        printf("CLASS   3   ERROR   0\n");
        printf(" ATG:  no error has occured\n");
        printf("SONY:  drive power is off\n");
        break;

        case 0x31:
        printf("CLASS   3   ERROR   1\n");
        printf(" ATG:  first blank condition has been met\n");
        printf("SONY:  no disk in drive\n");
        break;

        case 0x32:
        printf("CLASS   3   ERROR   2\n");
        printf(" ATG:  first blank condition is not met \n");
        printf("SONY:  drive is not ready\n");
        break;

        case 0x33:
        printf("CLASS   3   ERROR   3\n");
        printf(" ATG:  search data is unsucessful\n");
        printf("SONY:  no error has occured\n");
        break;

        default:
        err4(ecc);
        break;
        }
return;
}
        err4(ecc)
        int ecc;
        {
        switch(ecc){
        case 0x40:
        printf("CLASS   4   ERROR   0\n");
        printf(" ATG:  no error has occured\n");
        printf("SONY:  write warning, little alternate area\n");
        break;

        case 0x41:
        printf("CLASS   4   ERROR   1\n");
        printf(" ATG:  no error has occured\n");
        printf("SONY:  write error. disk is no longer valid for write\n");
```

```
    break;

    case 0x42:
    printf("CLASS   4   ERROR   2\n");
    printf(" ATG:  no error has occured\n");
    printf("SONY:  disk error. disk can no longer be used\n");
    break;

    case 0x43:
    printf("CLASS   4   ERROR   3\n");
    printf(" ATG:  no error has occured\n");
    printf("SONY:  cannot read disk id.\n");
    break;

    case 0x44:
    printf("CLASS   4   ERROR   4\n");
    printf(" ATG:  no error has occured\n");
    printf("SONY:  write protect error one\n");
    break;

    case 0x45:
    printf("CLASS   4   ERROR   5\n");
    printf(" ATG:  no error has occured\n");
    printf("SONY:  write protect error two\n");
    break;

    case 0x46:
    printf("CLASS   4   ERROR   6\n");
    printf(" ATG:  no error has occured\n");
    printf("SONY:  disk warning\n");
    break;

    case 0x47:
    printf("CLASS   4   ERROR   7\n");
    printf(" ATG:  no error has occured\n");
    printf("SONY:  alternation trouble\n");
    break;

    default:
    err5(ecc);
    break;
    }
return;
}

err5(ecc)
int ecc;
{
    switch(ecc){

    case 0x50:
    printf("CLASS   5   ERROR   0\n");
    printf(" ATG:  error in focusing servo\n");
    printf("SONY:  specified address not found\n");
    break;

    case 0x51:
    printf("CLASS   5   ERROR   1\n");
    printf(" ATG:  error in radial tracking servo\n");
    printf("SONY:  address block is not found\n");
    break;

    case 0x52:
    printf("CLASS   5   ERROR   2\n");
    printf(" ATG:  error in spining servo\n");
    printf("SONY:  no adresses can be read on this medium\n");
    break;

    case 0x53:
    printf("CLASS   5   ERROR   3\n");
    printf(" ATG:  initialization error\n");
    printf("SONY:  data could not be read\n");
    break;

    case 0x54:
    printf("CLASS   5   ERROR   4\n");
    printf(" ATG:  no error has occured\n");
    printf("SONY:  uncorrectable read error\n");
    break;

    case 0x55:
    printf("CLASS   5   ERROR   5\n");
    printf(" ATG:  out of RAM memory\n");
    printf("SONY:  tracking error \n");
    break;

    case 0x56:
    printf("CLASS   5   ERROR   6\n");
    printf(" ATG:  no error has occured\n");
    printf("SONY:  write servo error\n");
    break;
```

```
     case 0x57:
     printf("CLASS   5   ERROR   7\n");
     printf(" ATG:  no error has occured\n");
     printf("SONY:  write moniter error\n");
     break;

     case 0x58:
     printf("CLASS   5   ERROR   8\n");
     printf(" ATG:  no error has occurred\n");
     printf("SONY:  write verify error has occured\n");
     break;

     default:
     err6(ecc);
     break;
     }
return;
}


     err6(ecc)
     int ecc;
     {
     switch(ecc){
     case 0x60:
     printf("CLASS   6   ERROR   0\n");
     printf(" ATG:  no error has occured\n");
     printf("SONY:  no data at the specified address\n");
     break;

     case 0x61:
     printf("CLASS   6   ERROR   1\n");
     printf(" ATG:  no error has occured\n");
     printf("SONY:  blank check failure\n");
     break;

     case 0x62:
     printf("CLASS   6   ERROR   2\n");
     printf(" ATG:  no error has occured\n");
     printf("SONY:  controller diagnostic failure\n");
     break;

     case 0x63:
     printf("CLASS   6   ERROR   3\n");
     printf(" ATG:  no error has occured\n");
     printf("SONY:  drive diagnostic failure\n");
     break;

     case 0x64:
     printf("CLASS   6   ERROR   4\n");
     printf(" ATG:  no error has occured\n");
     printf("SONY:  diagnostics aborted\n");
     break;

     case 0x65:
     printf("CLASS   6   ERROR   5\n");
     printf(" ATG:  no error has occured\n");
     printf("SONY:  no data at the specified address\n");
     break;

     default:
     break;
     }
     return;
}



/*********************************************************************/
/* D I S P   B U F F E R   F R O M  D I S K   F U N C T I O N     */
/*********************************************************************/

displbu()
{
int h,i,m;
     h=0;
     i=0;
     m=0;
          printf("\n\n\n\n");
          while(h != cmdblk.buflen/16)
          {
          printf("%4X:  ",h*16);
          m=0;
                    while(m != 16){
                    printf("%2X ",buffer.by[i]);
                    m++;
                    i++;
                                   }
           printf("\n");
           h++;
           }
```

```
return;
}

/*********************************************************************/
/* L O A D   B U F F E R   F R O M  D I S K   F U N C T I O N     */
/*********************************************************************/

loadbuf(name,bulen,start)

char name[20];
int bulen,start;
{
     int add[200],e,h,i,m;
     FILE *in;

     if ( ( in=fopen(name,"r") ) != NULL)
          {
          e=0;
          h=start/16;
          i=start;
          while(i < bulen+start-1){
                    fscanf(in,"%4X:  ",&add[h]);
                    m=0;
                              while(m != 16){
                              fscanf(in,"%2X ",&buffer.by[i]);
                              m++;
                              i++;
                                        }
                     h++;
                              }
                         fclose(in);
          }
          else
          {
          e=0xff;
          }
     return(e);
}



/*********************************************************************/
/* M A K E   B U F F E R   C O P Y   O N   D I S K               */
/*********************************************************************/

makebuf(name)
char name[20];
{
     int add[200],buflen,e,h,i,m;
     FILE *in;
     if ( (in=fopen(name,"w")) != NULL)
     {
          e=0;
          i=0;
          h=0;
          while(h != cmdblk.buflen/16)
          {
          fprintf(in,"%4X:  ",h*16);
          m=0;
                              while(m != 16){
                    fprintf(in,"%2X ",buffer.by[i]);
                    m++;
                    i++;
                                        }
          fprintf(in,"\n");
          h++;
          }
          fclose(in);
     }
     else
          {
          e=0xff;
          }
return(e);
}


/*********************************************************************/
/* U S E R   D E F I N E D   B U F F E R   O P T I O N           */
/*********************************************************************/

userbuf(patlen,tralen,start)
int patlen,tralen,start;
{
int m,i,patr[127];
     i=1;
     while(i<=patlen){
     printf("** enter pattren element %d:  ",i);
     scanf("%X",&patr[i]);
     i++;
               }
```

```
     i=start;
     m=1;
     while (i<=tralen+start-1){
          buffer.by[i]=patr[m];
          if (m==patlen) {
                         m=1;
                         i++;
                         }
          else
                         {
                          i++;
                          m++;
                        }
                       }
return;
}


/*********************************************************************/
/* U S E R  D E F I N E D  B U F F E R  O P T I O N                   */
/*********************************************************************/

fillbuf(ele,tralen,start)
int ele,tralen,start;
{
int m,i,patlen;
     patlen=1;
     i=1;
     i=start;
     m=1;
     while (i<=tralen+start-1){
          buffer.by[i]=ele;
          if (m==patlen) {
                         m=1;
                         i++;
                         }
          else
                         {
                          i++;
                          m++;
                        }
                       }
return;
}
```

```
/*********************************************************************/
/* M A K E  B U F F E R  C O P Y  O N  P A P E R                     */
/*********************************************************************/



prinbuf()
{
     int add[200];
     FILE *in;
     int buflen,e,h,i,m;
     if ((in=fopen("PRN","w")) != NULL)
     {
     e=0;
     i=0;
     h=0;
     while(h != cmdblk.buflen/16)
     {
     fprintf(in,"%4X:  ",h*16);
     m=0;
                    while(m != 16){
                    fprintf(in,"%2X ",buffer.by[i]);
                    m++;
                    i++;
                    }
     fprintf(in,"\n");
     h++;
     }
     fclose(in);
     }
     else
     {
     e=0xff;
     }
return(e);
}

/*********************************************************************/
/* A S C I I  B U F F E R                                            */
/*********************************************************************/

ascibuf()
{
int h,i,m;
```

```
    h=0;
    i=0;
    m=0;
            printf("\n\n\n\n");
            while (h != cmdblk.buflen/16)
            {
            printf("%4X:   ", h*16);
            m=0;
                    while(m != 16){
                    if (buffer.by[i] < 0x31 || buffer.by[i] >0xff)
                    printf(". ");
                    else
            printf("%2c ",buffer.by[i]);
            m++;
            i++;
            }
    printf("\n");
    h++;
    }
    return;
}

/********************** USER GUIDE EXAMPLES ****************************/

    start()
    {
    clear();
    cmdblk.cdb[0]=0x1b;
    cmdblk.cdb[4]=1;
    cif(&cmdblk);
    return;
    }

    byter()
    {
    int i;
    i=0;
    while (i<= cmdblk.buflen)
            {
            printf("byte number%d....%X\n",i,buffer.by[i]);
            i++;
            }
    return;
    }

/***********************************************************************/
```

```
/* B U F N U M   D I S P L A Y                                         */
/***********************************************************************/

bufnum(start,stop)
int start,stop;
{
int i;
      i=start;
      while (i<stop)
              {
              printf("byte number%d...%x\n",i,buffer.by[i]);
              i++
              }
      return;
}
```

```
comment *
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+                                                                    +
+        NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY              +
+        -------- --------- -- --------- --- ----------              +
+                scsi command software for optical                   +
+             disk controlers meeting this standard                  +
+            [for use with the M.D.I. scsi-1 board ]                 +
+                  Command interface function                        +
+       S C S I    F U N C T I O N     L I B R A R Y                 +
+  developed by:  J.GORCZYCA and EDUARDO SANCHEZ VILLAGRAN           +
+                                                                    +
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
*
;
dgroup    group   _data,_bss
;
_text     segment byte public 'CODE'
          assume  cs:_text,ds:dgroup
          public _cif
;
ROMADD    DW 0009H
ROMSEG    DW 0D800H
;
_cif      proc near
            push bp
            mov  bp,sp
          push di
          push si
          push es
          push ds
          mov  si,[bp+4]
          call dword ptr ROMADD
          mov ax,0
          jnc home
          mov ax,0ffH
home:     pop  ds
          pop  es
          pop  si
          pop  di
          pop  bp
          ret
_cif      endp
;
_text     ends
;
_data     segment  word public 'data'
_data     ends
;
_bss      segment word public 'bss'
_bss      ends
          end
```

REFERENCES

4.1 REFERENCES

1) American National Standards Institute, publication X3.131-1986 Small Computer System Interface (SCSI).

2) Art Tech Gigadisc, GC 1001 OEM MANUAL,Ref:39 369 507, ATG Gigadisc (available from ATG, Woburn, MA).

3) IBM(1981), IBM Personal Computer Language Series, Macro Assembler by Microsoft.

4) Micro Design International, SCSI-1, SCSI HOST ADAPTER MANUAL (available for MDI, Winter Park, Florida).

5) Microsoft Corporation, Microsoft 'C': Microsoft C compiler, library reference manual, user guide and reference manual.

6) SONY corporation, WDC-2000-10/A Writable Disk Controller, Operating Instructions and Interface Manual.

7) Borland International Inc.(1987), Turbo C, Reference Guide.

8) Borland International Inc. (1987) Turbo C, User's Guide.

NBS-114A (REV. 2-8C)

U.S. DEPT. OF COMM. **BIBLIOGRAPHIC DATA SHEET** (See instructions)	1. PUBLICATION OR REPORT NO. NISTIR 89-4023	2. Performing Organ. Report No.	3. Publication Date JANUARY 1989

4. TITLE AND SUBTITLE

SCSI COMMAND SYSTEM
software support for the control of SCSI devices

5. AUTHOR(S)

John Gorczyca

6. PERFORMING ORGANIZATION (If joint or other than NBS, see instructions)

NATIONAL BUREAU OF STANDARDS
U.S. DEPARTMENT OF COMMERCE
GAITHERSBURG, MD 20899

7. Contract/Grant No.

8. Type of Report & Period Covered

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS (Street, City, State, ZIP)

10. SUPPLEMENTARY NOTES

☐ Document describes a computer program; SF-185, FIPS Software Summary, is attached.

11. ABSTRACT (A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here)

The SCSI Command System was created by NIST/NCTL personnel for the control of SCSI devices from a micro computer equipped with a SCSI host adapter. The Command Interfacing Function permits all SCSI standard and manufacturer unique commands to be sent to external devices. The system allows two levels of user programming. The upper and lower levels offer the ability to utilize libraries of commands, and the ability to edit system parameters and commands directly by using the system's variables. Programming for the system is done in the 'C' language. Also included with the documentation are references, they provide additional information that may be of reader interest.

12. KEY WORDS (Six to twelve entries; alphabetical order; capitalize only proper names; and separate key words by semicolons)

C language; Command Interfacing Function; Command Libraries; Host adapter; Lower level programing; Manufacturer Commands; Micro Computers; SCSI Devices; SCSI commands; System variables; Upper level programing

13. AVAILABILITY

☒ Unlimited

☐ For Official Distribution. Do Not Release to NTIS

☐ Order From Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402.

☒ Order From National Technical Information Service (NTIS), Springfield, VA. 22161

14. NO. OF PRINTED PAGES 127

15. Price $18.95

USCOMM-DC 6043-P80